SYBASE iAnywhere

SQL Anywhere® Server Database Administration

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Copyright © 2010 iAnywhere Solutions, Inc. Portions copyright © 2010 Sybase, Inc. All rights reserved.

This documentation is provided AS IS, without warranty or liability of any kind (unless provided by a separate written agreement between you and iAnywhere).

You may use, print, reproduce, and distribute this documentation (in whole or in part) subject to the following conditions: 1) you must retain this and all other proprietary notices, on all copies of the documentation or portions thereof, 2) you may not modify the documentation, 3) you may not do anything to indicate that you or anyone other than iAnywhere is the author or source of the documentation.

iAnywhere®, Sybase®, and the marks listed at http://www.sybase.com/detail?id=1011207 are trademarks of Sybase, Inc. or its subsidiaries. ® indicates registration in the United States of America.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Contents

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

About this book

This book describes how to run, manage, and configure SQL Anywhere databases. It describes database connections, the database server, database files, backup procedures, security, high availability, and administration utilities and options.

About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in four formats:

- **DocCommentXchange** DocCommentXchange is a community for accessing and discussing SQL Anywhere documentation on the web.

  To access the documentation, go to http://dcx.sybase.com.

- **HTML Help** On Windows platforms, the HTML Help contains the complete SQL Anywhere documentation, including the books and the context-sensitive help for SQL Anywhere tools.

  To access the documentation, choose **Start** » **Programs** » **SQL Anywhere 12** » **Documentation** » **HTML Help (English)**.

- **Eclipse** On Unix platforms, the complete Help is provided in Eclipse format. To access the documentation, run *sadoc* from the *bin32* or *bin64* directory of your SQL Anywhere installation.

- **PDF** The complete set of SQL Anywhere books is provided as a set of Portable Document Format (PDF) files. You must have a PDF reader to view information.

  To access the PDF documentation on Windows operating systems, choose **Start** » **Programs** » **SQL Anywhere 12** » **Documentation** » **PDF (English)**.

  To access the PDF documentation on Unix operating systems, use a web browser to open */documentation/en/pdf/index.html* under the SQL Anywhere installation directory.

Documentation conventions

This section lists the conventions used in this documentation.

Operating systems

SQL Anywhere runs on a variety of platforms. Typically, the behavior of the software is the same on all platforms, but there are variations or limitations. These are commonly based on the underlying operating system (Windows, Unix), and seldom on the particular variant (IBM AIX, Windows Mobile) or version.

To simplify references to operating systems, the documentation groups the supported operating systems as follows:

- **Windows** The Microsoft Windows family includes platforms that are used primarily on server, desktop, and laptop computers, as well as platforms used on mobile devices. Unless otherwise specified, when the documentation refers to Windows, it refers to all supported Windows-based platforms, including Windows Mobile.

  Windows Mobile is based on the Windows CE operating system, which is also used to build a variety of platforms other than Windows Mobile. Unless otherwise specified, when the documentation refers to Windows Mobile, it refers to all supported platforms built using Windows CE.

- **Unix** Unless otherwise specified, when the documentation refers to Unix, it refers to all supported Unix-based platforms, including Linux and Mac OS X.

For the complete list of platforms supported by SQL Anywhere, see "Supported platforms" [*SQL Anywhere 12 - Introduction*].

Directory and file names

Usually references to directory and file names are similar on all supported platforms, with simple transformations between the various forms. In these cases, Windows conventions are used. Where the details are more complex, the documentation shows all relevant forms.

These are the conventions used to simplify the documentation of directory and file names:

- **Uppercase and lowercase directory names** On Windows and Unix, directory and file names may contain uppercase and lowercase letters. When directories and files are created, the file system preserves letter case.

  On Windows, references to directories and files are *not* case sensitive. Mixed case directory and file names are common, but it is common to refer to them using all lowercase letters. The SQL Anywhere installation contains directories such as *Bin32* and *Documentation*.

  On Unix, references to directories and files *are* case sensitive. Mixed case directory and file names are not common. Most use all lowercase letters. The SQL Anywhere installation contains directories such as *bin32* and *documentation*.

  The documentation uses the Windows forms of directory names. You can usually convert a mixed case directory name to lowercase for the equivalent directory name on Unix.

- **Slashes separating directory and file names** The documentation uses backslashes as the directory separator. For example, the PDF form of the documentation is found in *install-dir* *\Documentation\en\PDF* (Windows form).

  On Unix, replace the backslash with the forward slash. The PDF documentation is found in *install-dir/documentation/en/pdf*.

- **Executable files** The documentation shows executable file names using Windows conventions, with a suffix such as *.exe* or *.bat*. On Unix, executable file names have no suffix.

  For example, on Windows, the network database server is *dbsrv12.exe*. On Unix, it is *dbsrv12*.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- ***install-dir*** During the installation process, you choose where to install SQL Anywhere. The environment variable SQLANY12 is created and refers to this location. The documentation refers to this location as *install-dir*.

  For example, the documentation may refer to the file *install-dir/readme.txt*. On Windows, this is equivalent to *%SQLANY12%\readme.txt*. On Unix, this is equivalent to *$SQLANY12/readme.txt* or *${SQLANY12}/readme.txt*.

  For more information about the default location of *install-dir*, see "SQLANY12 environment variable" on page 387.

- ***samples-dir*** During the installation process, you choose where to install the samples included with SQL Anywhere. The environment variable SQLANYSAMP12 is created and refers to this location. The documentation refers to this location as *samples-dir*.

  To open a Windows Explorer window in *samples-dir*, choose **Start** » **Programs** » **SQL Anywhere 12** » **Sample Applications And Projects**.

  For more information about the default location of *samples-dir*, see "SQLANYSAMP12 environment variable" on page 388.

Command prompts and command shell syntax

Most operating systems provide one or more methods of entering commands and parameters using a command shell or command prompt. Windows command prompts include Command Prompt (DOS prompt) and 4NT. Unix command shells include Korn shell and bash. Each shell has features that extend its capabilities beyond simple commands. These features are driven by special characters. The special characters and features vary from one shell to another. Incorrect use of these special characters often results in syntax errors or unexpected behavior.

The documentation provides command line examples in a generic form. If these examples contain characters that the shell considers special, the command may require modification for the specific shell. The modifications are beyond the scope of this documentation, but generally, use quotes around the parameters containing those characters or use an escape character before the special characters.

These are some examples of command line syntax that may vary between platforms:

- **Parentheses and curly braces** Some command line options require a parameter that accepts detailed value specifications in a list. The list is usually enclosed with parentheses or curly braces. The documentation uses parentheses. For example:

  ```
  -x tcpip(host=127.0.0.1)
  ```

  Where parentheses cause syntax problems, substitute curly braces:

  ```
  -x tcpip{host=127.0.0.1}
  ```

  If both forms result in syntax problems, the entire parameter should be enclosed in quotes as required by the shell:

  ```
  -x "tcpip(host=127.0.0.1)"
  ```

- **Semicolons** On Unix, semicolons should be enclosed in quotes.
- **Quotes** If you must specify quotes in a parameter value, the quotes may conflict with the traditional use of quotes to enclose the parameter. For example, to specify an encryption key whose value contains double-quotes, you might have to enclose the key in quotes and then escape the embedded quote:

  ```
  -ek "my \"secret\" key"
  ```

  In many shells, the value of the key would be `my "secret" key`.
- **Environment variables** The documentation refers to setting environment variables. In Windows shells, environment variables are specified using the syntax *%ENVVAR%*. In Unix shells, environment variables are specified using the syntax *$ENVVAR* or *${ENVVAR}*.

Contacting the documentation team

We would like to receive your opinions, suggestions, and feedback on this Help.

You can leave comments directly on help topics using DocCommentXchange. DocCommentXchange (DCX) is a community for accessing and discussing SQL Anywhere documentation. Use DocCommentXchange to:

- View documentation
- Check for clarifications users have made to sections of documentation
- Provide suggestions and corrections to improve documentation for all users in future releases

Go to http://dcx.sybase.com.

Finding out more and requesting technical support

Newsgroups

If you have questions or need help, you can post messages to the Sybase iAnywhere newsgroups listed below.

When you write to one of these newsgroups, always provide details about your problem, including the build number of your version of SQL Anywhere. You can find this information by running the following command: **dbeng12 -v**.

The newsgroups are located on the *forums.sybase.com* news server.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The newsgroups include the following:

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

For web development issues, see http://groups.google.com/group/sql-anywhere-web-development.

Newsgroup disclaimer
iAnywhere Solutions has no obligation to provide solutions, information, or ideas on its newsgroups, nor is iAnywhere Solutions obliged to provide anything other than a systems operator to monitor the service and ensure its operation and availability.

iAnywhere Technical Advisors, and other staff, assist on the newsgroup service when they have time. They offer their help on a volunteer basis and may not be available regularly to provide solutions and information. Their ability to help is based on their workload.

Developer Centers

The **SQL Anywhere Tech Corner** gives developers easy access to product technical documentation. You can browse technical white papers, FAQs, tech notes, downloads, techcasts and more to find answers to your questions as well as solutions to many common issues. See http://www.sybase.com/developer/library/sql-anywhere-techcorner.

The following table contains a list of the developer centers available for use on the SQL Anywhere Tech Corner:

Name	URL	Description
SQL Anywhere .NET Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/microsoft-net	Get started and get answers to specific questions regarding SQL Anywhere and .NET development.
PHP Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/php	An introduction to using the PHP (PHP Hypertext Preprocessor) scripting language to query your SQL Anywhere database.

Name	URL	Description
SQL Anywhere Windows Mobile Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/windows-mobile	Get started and get answers to specific questions regarding SQL Anywhere and Windows Mobile development.

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Starting and connecting to your database

This section describes how to start the SQL Anywhere database server, and how to connect to your database from a client application.

Tutorial: Using the sample database

This tutorial focuses on the sample database. The sample database represents a small company that makes a limited range of sports clothing. It contains internal information about the company (employees, departments, and financial data), product information (products) and sales information (sales orders, customers, and contacts). All information in the sample database is fictional. See "SQL Anywhere sample database" [*SQL Anywhere 12 - Introduction*].

Lesson 1: Make a copy of the sample database

Before you begin, make a copy of the sample database so that you can restore it after you have made changes.

To copy the sample database

1. Create a directory to hold the copy of the sample database you will use in this tutorial, for example *c:\demodb*.
2. Run the following command to create a database named *demo.db* that contains data from the sample database:

   ```
   newdemo c:\demodb\demo.db
   ```

Lesson 2: Start the SQL Anywhere database server

To start a personal database server running the sample database (command line)

- Run the following command to start the personal database server, name the database server mydemo12 using the -n server option, and connect to the copy of the sample database:

  ```
  dbeng12 -n mydemo12 c:\demodb\demo.db
  ```

  On Windows, the database server appears as an icon in the system tray.

For more information about starting the network database server, see "Start the database server" on page 34.

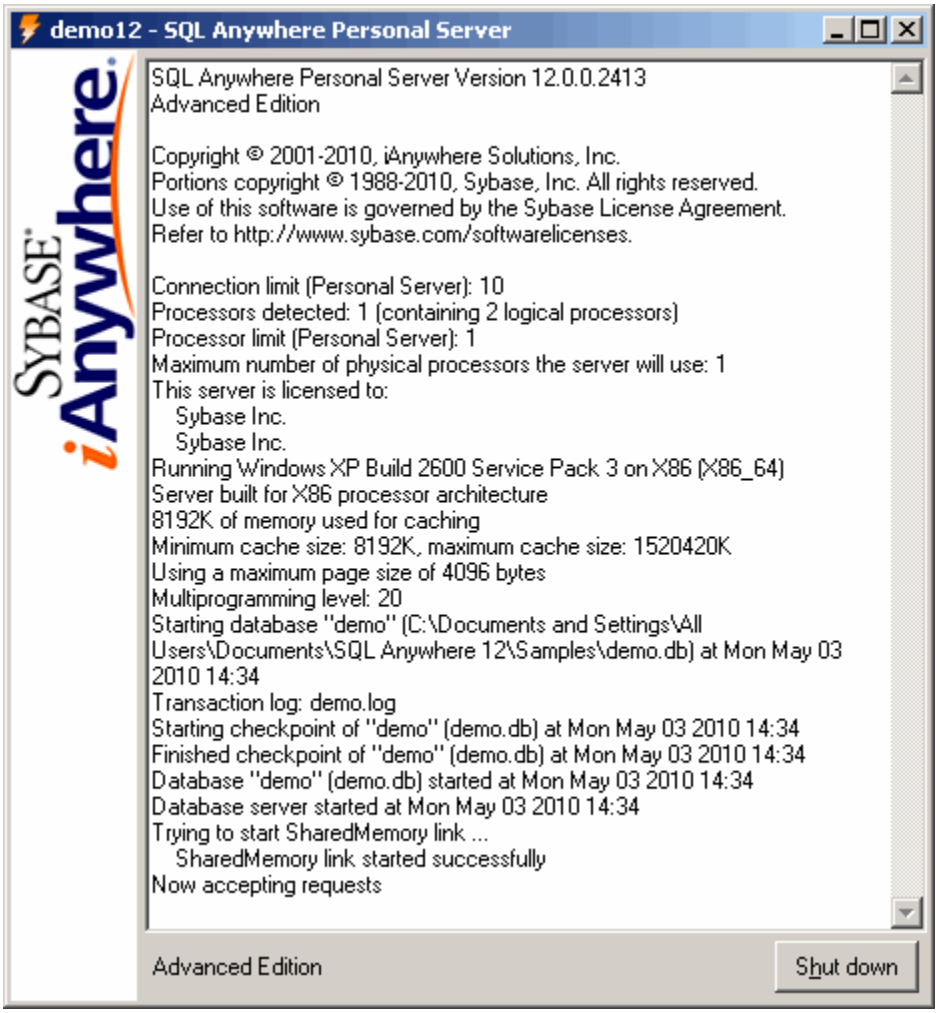
See also

- "Using SQL Anywhere database servers" on page 33
- "The SQL Anywhere database server" on page 147

Lesson 3: Display the database server messages window

You have successfully started a personal database server running the sample database. However, you cannot see or manipulate the data in the database yet.

The SQL Anywhere personal server icon is the only visible indication that anything has happened. You can display the database server messages window in Windows by double-clicking the SQL Anywhere personal server icon in the system tray.



The database server messages window displays useful information, including:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **The server name** The name in the title bar (in this case mydemo12) is the **server name**. In this tutorial, you assigned the server name using the -n server option. If you don't provide a server name, the database server is given the name of the first database started. This name can be used by applications when they connect to a database. See "Naming the database server and the databases" on page 39.

- **The version and build numbers** The numbers following the server name (for example, **12.0.0.2413**) are the version and build numbers. The version number represents the specific release of SQL Anywhere, and the build number relates to the specific instance of the software that was compiled.

- **Startup information** When a database server starts, it sets aside some memory that it uses when processing database requests. This reserved memory is called the **cache**. The amount of cache memory appears in the window. The cache is organized in fixed-size **pages**, and the page size also appears in the window.

- **Database information** The names of the database file and its transaction log file appear in the window.

  In this case, the startup cache size and page size are the default values. For many purposes, including those of this tutorial, the default startup options are fine.

Lesson 4: Stop the database server

You can now stop the database server you just started.

In Windows, you can stop a database server by clicking Shut Down on the database server messages window.

To stop the database server running the sample database (Windows)

1. Double-click the SQL Anywhere icon in the system tray.
2. Click **Shut Down**.

To stop the database server running the sample database (command line)

- Run the following command to stop the personal database server running the sample database:

  ```
  dbstop mydemo12
  ```

  The Stop Server utility (dbstop) can only be run at a command prompt. See "Stop Server utility (dbstop)" on page 851.

Tutorial cleanup

Once you have shut down the database server, you can delete the *c:\demodb* directory and its contents.

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Starting Interactive SQL" on page 698
- "Working with the Connect window" on page 92
- "Using SQL Anywhere database servers" on page 33

Working with database files

Each database has the following files associated with it:

- **The database file** This file holds the database information. It typically has the extension *.db*.
- **The transaction log** This file holds a record of the changes made to the database, and is necessary for recovery and synchronization. It typically has the extension *.log*. See "The transaction log" on page 21.
- **The temporary file** The database server uses the temporary file to hold information needed during a database session. The database server discards this file once the database shuts down—even if the server remains running. The file has a server-generated name with the extension *.tmp*.

  The location of the temporary file can be specified when starting the database server using the -dt server option. If you do not specify the location of the temporary file when starting the database server, the following environment variables are checked, in order:

  - SATMP environment variable
  - TMP environment variable
  - TMPDIR environment variable
  - TEMP environment variable

  If none of these environment variables are defined, SQL Anywhere places its temporary file in the current directory on Windows operating systems, or in the */tmp* directory on Unix.

  The database server creates, maintains, and removes the temporary file. You only need to ensure that there is enough free space available for the temporary file. You can obtain information about the space available for the temporary file using the sa_disk_free_space procedure. See "sa_disk_free_space system procedure" [*SQL Anywhere Server - SQL Reference*].
- **Predefined dbspace files** These files store your data and other files used by the database. See "Predefined dbspaces" on page 15.

Additional files

Other files can also be part of a database system, including:

- **Dbspace files** You can spread your data over several separate files, in addition to the database file. See "Using additional dbspaces" on page 16.
- **Transaction log mirror files** For additional security, you can create a mirror copy of the transaction log. This file typically has the extension *.mlg*. See "Transaction log mirrors" on page 22.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Creating a SQL Anywhere database

To create a SQL Anywhere database, you first define the tables it will have (entities), the columns in each table (attributes), and the relationships between tables (keys and constraints).

It is recommended that you create a Conceptual Database Model (CDM) for your new database. You can use CDM applications such as Sybase PowerDesigner to create and validate your database design. These tools construct SQL statements that are submitted to the database server, typically through its ODBC interface. If you are using one of these tools, you do not need to construct SQL statements to create tables, assign permissions, and so on. See "PowerDesigner Physical Data Model" [*SQL Anywhere 12 - Introduction*].

For more information about designing databases and creating a CDM, see the Sybase PowerDesigner documentation at http://infocenter.sybase.com/help/index.jsp.

For more information about database objects such as tables and views, see "Working with database objects" [*SQL Anywhere Server - SQL Usage*].

You can also use Sybase Central, Interactive SQL, or the command line to create or **initialize** a SQL Anywhere database. After creating the database, you can connect to it and add tables and other objects.

Transaction log

When you create a database, you must decide where to place the transaction log. This log stores all changes made to a database, in the order in which they are made. In the event of a media failure on a database file, the transaction log is essential for database recovery. It also makes your work more efficient. By default, it is placed in the same directory as the database file, but this configuration is not recommended for production use.

For more information about placing the transaction log, see "The transaction log" on page 21.

Database file compatibility

A SQL Anywhere database is an operating system file. It can be copied to other locations just as any other file is copied.

Database files are compatible among all operating systems, except where file system file size limitations or SQL Anywhere support for large files apply. See "SQL Anywhere size and number limitations" on page 675.

A database created from any operating system can be used from another operating system by copying the database file(s). Similarly, a database created with a personal database server can be used with a network database server. SQL Anywhere database servers can manage databases created with earlier versions of the software, but old servers cannot manage newer databases.

See also

- "Viewing entity-relationship diagrams from the SQL Anywhere 12 plug-in" on page 693

Choosing object names

Do not use reserved words to name database objects. For a list of SQL Anywhere reserved words, see "Reserved words" [*SQL Anywhere Server - SQL Reference*].

Column names must be enclosed in double quotes if they contain characters other than letters, numbers, or underscores, if they do not begin with a letter, or if the name is the same as a keyword.

Choosing column data types

The following data types are available in SQL Anywhere:

- Binary data types
- Character data types
- Date/time data types
- Decimal data types
- Domains (user-defined data types)
- Floating-point data types
- Integer data types
- Spatial data types

For more information about data types, see "SQL data types" [*SQL Anywhere Server - SQL Reference*].

Any of the character or binary string data types such as CHAR, VARCHAR, LONG VARCHAR, NCHAR, BINARY, VARBINARY, and so on, can be used to store large objects such as images, word-processing documents, and sound files.

For more information about BLOB storage, see "Storing BLOBs" on page 7.

NULL and NOT NULL columns

If the column value is mandatory for a row, you define the column as being NOT NULL. Otherwise, the column is allowed to contain the NULL value, which represents no value. The default in SQL Anywhere is to allow NULL values, but you should explicitly declare columns NOT NULL unless there is a good reason to allow NULL values.

The SQL Anywhere sample database has a table called Departments, which has columns named DepartmentID, DepartmentName, and DepartmentHeadID. Its definition is as follows:

Column	Data type	Size	Null/not null	Constraint
DepartmentID	integer	—	NOT NULL	None
DepartmentName	char	40	NOT NULL	None

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Column	Data type	Size	Null/not null	Constraint
DepartmentHeadID	integer	—	NULL	None

If you specify NOT NULL, a column value must be supplied for every row in the table.

For more information about the NULL value, see "NULL value" [*SQL Anywhere Server - SQL Reference*]. For information about its use in comparisons, see "Search conditions" [*SQL Anywhere Server - SQL Reference*].

Storing BLOBs

A BLOB is an uninterpreted string of bytes or characters, stored as a value in a column. Common examples of a BLOB are picture or sound files. While BLOBs are typically large, you can store them in any character string or binary string data type such as CHAR, VARCHAR, NCHAR, BINARY, VARBINARY, and so on. Choose your data type and length depending on the content and length of BLOBs you expect to store.

Note
While a character large object is commonly called a CLOB, a binary large object is called a BLOB, and the combination of both is called a LOB. Only the acronym BLOB is used in this documentation.

When you create a column for storing BLOB values, you can control aspects of their storage. For example, you can specify that BLOBs up to a specified size be stored in the row (inline), while larger BLOBs are stored outside the row in table extension pages. Additionally, you can specify that for BLOBs stored outside the row, the first *n* bytes of the BLOB, also referred to as the prefix, are duplicated in the row. These storage aspects are controlled by the INLINE and PREFIX settings specified in the CREATE TABLE and ALTER TABLE statements. The values you specify for these settings can have unanticipated impacts on performance or disk storage requirements.

If neither INLINE nor PREFIX is specified, or if INLINE USE DEFAULT or PREFIX USE DEFAULT is specified, default values are applied as follows:

- For character data type columns, such as CHAR, NCHAR, LONG VARCHAR, and XML, the default value of INLINE is 256, and the default value of PREFIX is 8.
- For binary data type columns, such as BINARY, LONG BINARY, VARBINARY, BIT, VARBIT, LONG VARBIT, BIT VARYING, and UUID, the default value of INLINE is 256, and the default value of PREFIX is 0.

It is recommended that you do not set INLINE and PREFIX values unless there are specific requirements for which the default values are insufficient. The default values have been chosen to balance performance and disk space requirements. For example, row processing performance may degrade if you set INLINE to a large value, and all the BLOBs are stored inline. If you set PREFIX too high, you increase the amount of disk space required to store BLOBs since the prefix data duplicates a portion of the BLOB.

If you do decide to set INLINE or PREFIX values, the INLINE length must not exceed the length of the column. Likewise, the PREFIX length, must not exceed the INLINE length.

The prefix data for a compressed column is stored uncompressed, so if all the data required to satisfy a request is stored in the prefix, no decompression is necessary.

For information about the defaults for the INLINE and PREFIX clauses, see "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*].

BLOB sharing

If a BLOB exceeds the inline size, and requires more than one database page for storage, the database server stores it so that it can be referenced by other rows in the same table, when possible. This is known as BLOB sharing. BLOB sharing is handled internally and is intended to reduce unnecessary duplication of BLOBs in the database.

BLOB sharing only occurs when you set values of one column to be equal to those of another column. For example, `UPDATE t column1=column2;`. In this example, if column2 contains BLOBs, instead of duplicating them in column1, pointers to the values in column2 are used instead.

When a BLOB is shared, the database server keeps track of how many other references there are to the BLOB. Once the database server determines that a BLOB is no longer referenced within a table, the BLOB is removed.

If a BLOB is shared between two uncompressed columns and one of those columns is then compressed, the BLOB will no longer be shared.

Choosing column compression

CHAR, VARCHAR, and BINARY columns can be compressed to save disk space. For example, you can compress a column in which large BLOB files such as BMPs and TIFFs are stored. Compression is achieved using the deflate compression algorithm. This is the same algorithm used by the COMPRESS function, and is also the same algorithm used for Windows ZIP files.

Compressed columns can reside inside of encrypted tables. In this case, data is first compressed, and then encrypted.

Do not use column compression on columns containing values under 130 bytes, or values that are already in a compressed format, such as JPG files. Attempting to compress columns that contain values that are already compressed may actually increase the amount of storage required for the column.

To compress columns, use the COMPRESS clause of the CREATE TABLE and ALTER TABLE statements.

You can determine the benefits you are getting by compressing columns using the sa_column_stats system procedure.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "ALTER TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "sa_column_stats system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Table encryption" on page 1139

Choosing constraints

Although the data type of a column restricts the values that are allowed in that column (for example, only numbers or dates), you may want to further restrict the allowed values.

You can restrict the values of any column by specifying a CHECK constraint. A CHECK constraint is a restriction that enforces specified conditions on a column or set of columns. You can use any valid condition that could appear in a WHERE clause to restrict the allowed values. Most CHECK constraints use either the BETWEEN or IN condition.

For more information about valid conditions, see "Search conditions" [*SQL Anywhere Server - SQL Reference*]. For more information about assigning constraints to tables and columns, see "Ensuring data integrity" [*SQL Anywhere Server - SQL Usage*].

Create a database (Sybase Central)

You can create a database in Sybase Central using the **Create Database Wizard**. See "Create a database (SQL)" on page 9, and "Create a database (command line)" on page 10.

To create a new database (Sybase Central)

1. Choose **Tools** » **SQL Anywhere 12** » **Create Database**.
2. Follow the instructions in the **Create Database Wizard**.

Tip
You can also access the **Create Database Wizard** from within Sybase Central using the following methods:

- Selecting a server, and choosing **File** » **Create Database**.
- Right-clicking a server, and choosing **Create Database**.

Creating databases for Windows Mobile

For information about creating databases for Windows Mobile, see "Creating a Windows Mobile database" on page 355.

Create a database (SQL)

In Interactive SQL, use the CREATE DATABASE statement to create databases. You need to connect to an existing database before you can use this statement.

To create a new database (SQL)

1. Start a database server named sample.

   ```
   dbeng12 -n sample
   ```

2. Start Interactive SQL.

3. Connect to an existing database. If you don't have a database, you can connect to the utility database utility_db. See "Connecting to the utility database" on page 29.

4. Execute a CREATE DATABASE statement.

See "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

Example

Create a database file in the *c:\temp* directory with the file name *temp.db*.

```
CREATE DATABASE 'c:\\temp\\temp.db';
```

The directory path is relative to the database server. You set the permissions required to execute this statement on the server command line, using the -gu option. The default setting requires DBA authority.

The backslash is an escape character in SQL, and sometimes must be doubled. The \x and \n sequences can be used to specifying hexadecimal and newline characters. Letters other than n and x do not have any special meaning if they are preceded by a backslash. Here are some examples where this is important.

```
CREATE DATABASE 'c:\\temp\\\x41\x42\x43xyz.db';
```

The initial \\ sequence represents a backslash. The \x sequences represent the characters A, B, and C, respectively. The file name here is *ABCxyz.db*.

```
CREATE DATABASE 'c:\temp\\nest.db';
```

To avoid having the \n sequence interpreted as a newline character, the backslash is doubled.

See "Escape sequences" [*SQL Anywhere Server - SQL Reference*].

Create a database (command line)

You can create a database from a command line with the Initialization utility (dbinit). With this utility, you can include command line options to specify different settings for the database.

To create a new database (command line)

- Run a dbinit command.

  For example, to create a database called *company.db* with a 4 KB page size, run the following command:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
dbinit -p 4k company.db
```

See also

- "Initialization utility (dbinit)" on page 799

Create a database with a transaction log mirror

You can choose to maintain a transaction log mirror when you create a database. This option is available from the CREATE DATABASE statement, from Sybase Central, or from the dbinit utility.

For more information about why you may want to use a transaction log mirror, see "Transaction log mirrors" on page 22.

To create a database that uses a transaction log mirror (Sybase Central)

1. From the **Tools** menu, choose **SQL Anywhere 12** » **Create Database**.
2. Follow the instructions in the **Create Database Wizard**.

To create a database that uses a transaction log mirror (SQL)

- Use the CREATE DATABASE statement, with the TRANSACTION LOG and MIRROR clauses. For example:

  ```
  CREATE DATABASE 'c:\\mydb'
  TRANSACTION LOG ON mydb.log
  MIRROR 'd:\\mydb.mlg';
  ```

  See "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

To create a database that uses a transaction log mirror (command line)

- Use the dbinit utility with the -m option. For example, the following command (which should be entered on one line) initializes a database named *company.db*, with a transaction log kept on a different device and a mirror on a third device.

  ```
  dbinit -t d:\log-dir\company.log -m
  e:\mirr-dir\company.mlg c:\db-dir\company.db
  ```

  See "Initialization utility (dbinit)" on page 799.

Tutorial: Creating a SQL Anywhere database

This tutorial describes how to use Sybase Central to create a simple database, modeled on the Products, SalesOrderItems, SalesOrders, and Customers tables of the SQL Anywhere sample database.

For information about the SQL Anywhere sample database, see "SQL Anywhere sample database" [*SQL Anywhere 12 - Introduction*].

Lesson 1: Create a database file

Use the following procedure to create a database file to hold your database. The database file contains system tables and other system objects that are common to all databases.

To create a new database file

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. Choose **Tools** » **SQL Anywhere 12** » **Create Database**.
3. On the **Welcome** page click **Next**.
4. Select **Create A Database On This Computer**, and then click **Next**.
5. In the **Save The Main Database File To The Following File** field, type *c:\temp\mysample.db*.

   If your temporary directory is somewhere other than *c:\temp*, specify the appropriate path.
6. Click **Finish**.
7. Click **Close**.

See also

- "Lesson 2: Add tables to the database" on page 12
- "Lesson 3: Set a NOT NULL constraint on a column" on page 14
- "Lesson 4: Create a foreign key" on page 14

Lesson 2: Add tables to the database

Use the following procedure to create a table named Products.

To create a table

1. In the right pane of Sybase Central, double-click **Tables**.
2. Right-click **Tables** and choose **New** » **Table**.
3. In the **What Do You Want To Name The New Table** field, type **Products**.
4. Click **Finish**.

   The database server creates the table using defaults, and then displays the **Columns** tab in the right pane. The **Name** field for the new column is selected and a prompt waits for you to specify a name for the new column.
5. Type **ProductID** as the name for the new column.

   Since this is the first column in the table, **PKey** is selected, indicating that the column is the primary key for the table.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

When creating a table, you can create a primary key that is made up of more than one columns by creating the columns and placing a checkmark in each **PKey** column. See "Primary keys" [*SQL Anywhere 12 - Introduction*].

6. In the **Data Type** list, select **Integer**.
7. Click the ellipsis (three dots) button.
8. Click the **Value** tab and choose **Default Value** » **System-Defined** » **Autoincrement**.

   An autoincrement value increments for each row added to the table. This ensures that values in the column are unique—a requirement for primary keys. See "Primary keys" [*SQL Anywhere 12 - Introduction*].
9. Click **OK**.
10. From the **File** menu, choose **New** » **Column**.
11. Complete the following fields:
    - In the **Name** field, type **ProductName**.
    - In the **Data Type** list, select **Char**.
    - In the **Size** list, select **15**.
12. Add the following tables to your database.
    - **Customers table** Add a table named **Customers** with the following columns:
      - **CustomersID** An identification number for each customer. Make sure **PKey** is selected, set the **Data Type** to **Integer**, and set the **Default Value** to **Autoincrement**.
      - **CompanyName** The name of each company. Set the **DataType** to **Char** with a maximum length of **35** characters.
    - **SalesOrders table** Add a table named **SalesOrders** with the following columns:
      - **SalesOrdersID** An identification number for each sales order. Set the **DataType** to **Integer**, and make sure **PKey** is selected. Set the **Default Value** to **Autoincrement**.
      - **OrderDate** The date on which the order was placed. Set the **DataType** to **Date**.
      - **CustomerID** The identification number of the customer who placed the sales order. Set the **DataType** to **Integer**.
    - **SalesOrderItems table** Add a table named **SalesOrderItems** with the following columns:
      - **SalesOrderItemsID** The identification number of the sales order of which the item is a part. Set the **DataType** to **Integer**, and make sure **PKey** is selected.
      - **LineID** An identification number for each sales order. Set the **DataType** to **Integer**, and make sure **PKey** is selected.

      Note
      Since **PKey** is set for both SalesOrderItemsID and LineID, this means the primary key for the table comprises the concatenated values of these two columns.

- **ProductID** The identification number for the product being ordered. Set the **DataType** to **Integer**.

13. From the **File** menu, choose **Save**.

See also

- "Lesson 1: Create a database file" on page 12
- "Lesson 3: Set a NOT NULL constraint on a column" on page 14
- "Lesson 4: Create a foreign key" on page 14
- "Viewing entity-relationship diagrams from the SQL Anywhere 12 plug-in" on page 693

Lesson 3: Set a NOT NULL constraint on a column

Use the following procedure to add a NOT NULL constraint to a column.

To add and remove a constraint on a column

1. In the left pane of Sybase Central, double-click **Tables**.
2. Click **Products**, and then click the **Columns** tab in the right pane.
3. Select the **ProductName** column.
4. From the **File** menu, choose **Properties**.
5. Click the **Constraints** tab and select **Values Cannot Be NULL**.

   By default, columns allow NULLs, but it is good practice to declare columns NOT NULL unless there is a good reason to allow NULLs. See "NULL value" [*SQL Anywhere Server - SQL Reference*].
6. Click **OK**.

   This constraint means that for each row added to the **Products** table, the **ProductName** column must have a value.
7. From the **File** menu, choose **Save**.

See also

- "Lesson 1: Create a database file" on page 12
- "Lesson 2: Add tables to the database" on page 12
- "Lesson 4: Create a foreign key" on page 14

Lesson 4: Create a foreign key

Use the following procedure to use foreign keys to create relationships between tables.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To create a foreign key

1. In the left pane of Sybase Central, double-click **Tables**.
2. In the left pane, click the **SalesOrdersItems** table to select it.
3. In the right pane, select the **Constraints** tab.
4. Choose **File** » **New** » **Foreign Key**.
5. In the **To Which Table Do You Want This Foreign Key To Refer** list, select the **Products** table.
6. In the **What Do You Want To Name The New Foreign Key** field, type **ProductIDkey**.
7. Click **Next** and for **Do You Want This Foreign Key To Reference The Primary Key Or A Unique Constraint** choose **Primary Key**.
8. In the **Foreign Column** list, click **SalesOrdersItemsID**.
9. Click **Finish**.

See also

- "Lesson 1: Create a database file" on page 12
- "Lesson 2: Add tables to the database" on page 12
- "Lesson 3: Set a NOT NULL constraint on a column" on page 14

Predefined dbspaces

SQL Anywhere uses the following predefined dbspaces for its databases:

Dbspace	Name
Main database file	system
Temporary file	temporary or temp
Transaction log file	translog
Transaction log mirror	translogmirror

You cannot create user-defined dbspaces with these names and you cannot drop the predefined dbspaces.

If you upgrade a version 10.0.0 or earlier database with user-defined dbspaces that use the predefined dbspace names, then all references to these dbspaces in SQL statements are assumed to be referring to the user-defined dbspaces, and not the predefined dbspaces. The only way that you can refer to the predefined dbspaces is by dropping the user-defined dbspaces, or renaming them to not use the same names as the predefined dbspaces.

The ALTER DBSPACE statement supports the predefined dbspace names so you can add more space to them. See "ALTER DBSPACE statement" [*SQL Anywhere Server - SQL Reference*].

The DB_EXTENDED_PROPERTY function also accepts the predefined dbspace names. See "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].

Using additional dbspaces

Typically needed for large databases
For most databases, a single database file is enough. However, for users of large databases, additional database files are sometimes necessary. Additional database files are also convenient tools for clustering related information in separate files.

When you initialize a database, it contains one database file. This first database file is called the **main file** or the **system** dbspace. By default, all database objects and all data are placed in the main file.

A **dbspace** is an additional database file that creates more space for data. A database can be held in up to 13 separate files (the main file and 12 dbspaces). Each table, together with its indexes, must be contained in a single database file. The SQL command CREATE DBSPACE adds a new file to the database.

Temporary tables are only created in the temporary dbspace.

There are several ways to specify the dbspace where a base table or other database object is created. In the following lists, the location specified by methods occurring earlier in the list take precedence over those occurring later in the list.

1. IN DBSPACE clause (if specified)
2. default_dbspace option (if set)
3. system dbspace

If a dbspace name contains a period and is not quoted, the database server generates an error for the name.

Each database file has a maximum allowable size of 2^{28} (approximately 268 million) database pages. For example, a database file created with a database page size of 4 KB can grow to a maximum size of one terabyte (2^{28}*4 KB). However, in practice, the maximum file size allowed by the physical file system in which the file is created affects the maximum allowable size significantly.

While some older file systems restrict file size to a maximum of 2 GB, many file systems, such as Windows using the NTFS file system, allow you to exploit the full database file size. In scenarios where the amount of data placed in the database exceeds the maximum file size, it is necessary to divide the data into more than one database file. As well, you may want to create multiple dbspaces for reasons other than size limitations, for example, to cluster related objects.

For information about the maximum file size allowed on the supported operating systems, see "SQL Anywhere size and number limitations" on page 675.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can use the sa_disk_free system procedure to obtain information about space available for a dbspace. See "sa_disk_free_space system procedure" [*SQL Anywhere Server - SQL Reference*].

The SYSDBSPACE system view contains information about all the dbspaces for a database. See "SYSDBSPACE system view" [*SQL Anywhere Server - SQL Reference*].

Splitting existing databases

If you want to split existing database objects among multiple dbspaces, you must unload your database and modify the generated command file (named *reload.sql* by default) for rebuilding the database. In the *reload.sql* file, add IN clauses to the CREATE TABLE statements to specify the dbspace for each table you do not want to place in the main file.

Permissions on dbspaces

Only the CREATE permission is supported on dbspaces. The CREATE permission allows a user to create database objects in the specified dbspace. You can grant CREATE permission for a dbspace by executing a GRANT CREATE ON statement. See "GRANT statement" [*SQL Anywhere Server - SQL Reference*].

Dbspace permissions behave as follows:

- A user trying to create a new object with underlying data must have CREATE permission on the dbspace where the data is being placed.
- Even if a GRANT CREATE ON statement was issued, the user (grantee) must have RESOURCE authority to create new database objects.
- The current list of objects that can be placed in specific dbspaces, and that require the CREATE permission, includes tables, indexes, text indexes, and materialized views. Note that objects such as normal views and procedures do not have any underlying data and do not require the CREATE permission.
- A user can be granted the CREATE permission directly, or they can inherit the permission through membership in a group that has been granted the permission.
- It is possible to grant PUBLIC the CREATE permission on a specific dbspace, in which case any user who also has RESOURCE authority can create objects on the dbspace.
- A newly-created dbspace automatically grants CREATE permission on itself to PUBLIC.
- It is possible to revoke permissions, for example when trying to secure a dbspace. Permissions on the internal dbspaces system and temporary can also be managed to control access.
- Creating local temporary tables does not require any permissions; dbspace permissions do not affect the creation of local temporary tables. However, the creation of global temporary tables requires RESOURCE authority and CREATE permission on the temporary dbspace.

See also

- "CREATE DBSPACE statement" [*SQL Anywhere Server - SQL Reference*]
- "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "UNLOAD statement" [*SQL Anywhere Server - SQL Reference*]

Creating dbspaces

You create a new database file, or dbspace, either from Sybase Central, or using the CREATE DBSPACE statement. The database file for a new dbspace can be located on the same disk drive as the main file or on another disk drive. You must have DBA authority to create dbspaces.

For each database, you can create up to twelve dbspaces in addition to the main dbspace. A newly-created dbspace is empty. When you create a new table or index you can place it in a specific dbspace with an IN clause in the CREATE statement or set the default_dbspace option before creating the table. If you don't specify an IN clause, and don't change the setting of the default_dbspace option, the table is created in the system dbspace.

Each table is contained entirely in the dbspace it is created in. By default, indexes appear in the same dbspace as their table, but you can place them in a separate dbspace by supplying an IN clause as part of the CREATE statement.

See also

- "default_dbspace option" on page 532
- "CREATE DBSPACE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE INDEX statement" [*SQL Anywhere Server - SQL Reference*]

Create a dbspace

To create a dbspace (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database.
2. Open the **Dbspaces** folder for the database.
3. Choose **File** » **New** » **Dbspace**.
4. Follow the instructions in the **Create Dbspace Wizard**.

   The new dbspace appears in the **Dbspaces** folder.

To create a dbspace (SQL)

- Execute a CREATE DBSPACE statement.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Examples

The following command creates a new dbspace called MyLibrary in the file *library.db* in the same directory as the main file:

```
CREATE DBSPACE MyLibrary
AS 'library.db';
```

The following command creates a table LibraryBooks and places it in the MyLibrary dbspace.

```
CREATE TABLE LibraryBooks (
title CHAR(100),
author CHAR(50),
isbn CHAR(30)
) IN MyLibrary;
```

The following commands create a new dbspace named MyLibrary, set the default dbspace to the MyLibrary dbspace, and then create the LibraryBooks table in the MyLibrary dbspace.

```
CREATE DBSPACE MyLibrary
AS 'e:\\dbfiles\\library.db';
SET OPTION default_dbspace = 'MyLibrary';
CREATE TABLE LibraryBooks (
  title CHAR(100),
  author CHAR(50),
  isbn CHAR(30),
);
```

See also

- "CREATE DBSPACE statement" [*SQL Anywhere Server - SQL Reference*]
- "default_dbspace option" on page 532
- "Working with tables" [*SQL Anywhere Server - SQL Usage*]
- "CREATE INDEX statement" [*SQL Anywhere Server - SQL Reference*]

Preallocating space for database files

When you create a new database file, you can preallocate database space using the DATABASE SIZE clause of the CREATE DATABASE statement or by specifying the dbinit -dbs option. See "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*], and "Initialization utility (dbinit)" on page 799.

As you use the database, SQL Anywhere automatically grows database files as needed. Rapidly-changing database files can lead to excessive file fragmentation on the disk, resulting in potential performance problems. As well, many small allocations are slower than one large allocation. If you are working with a database with a high rate of change, you can preallocate disk space for dbspaces or for transaction logs using either Sybase Central or the ALTER DBSPACE statement. See "ALTER DBSPACE statement" [*SQL Anywhere Server - SQL Reference*].

You must have DBA authority to alter the properties of a database file.

Performance tip
Running a disk defragmentation utility after preallocating disk space helps ensure that the database file is not fragmented over many disjointed areas of the disk drive. Performance can suffer if there is excessive fragmentation of database files.

To preallocate space (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Open the **Dbspaces** folder.
3. Right-click the dbspace and choose **Preallocate Space**.
4. Enter the amount of space to add to the dbspace. You can add space in units of pages, bytes, kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB).
5. Click **OK**.

To preallocate space (SQL)

1. Connect to a database.
2. Execute an ALTER DBSPACE statement.

Examples

Increase the size of the system dbspace by 200 pages.

```
ALTER DBSPACE system
ADD 200;
```

Increase the size of the system dbspace by 400 megabytes.

```
ALTER DBSPACE system
ADD 400 MB;
```

See also

- "Creating dbspaces" on page 18
- "ALTER DBSPACE statement" [*SQL Anywhere Server - SQL Reference*]

Delete a dbspace

You can delete a dbspace using either Sybase Central or the DROP DBSPACE statement. Before you can delete a dbspace, you must delete all tables and indexes that use the dbspace. You must have DBA authority to delete a dbspace.

To delete a dbspace (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

2. Open the **Dbspaces** folder.

3. Right-click the dbspace and choose **Delete**.

To delete a dbspace (SQL)

1. Connect to a database.

2. Execute a DROP DBSPACE statement.

See also

- "Dropping tables" [*SQL Anywhere Server - SQL Usage*]
- "DROP DBSPACE statement" [*SQL Anywhere Server - SQL Reference*]

The transaction log

The **transaction log** is a separate file from the database file. It stores all changes to the database. Inserts, updates, deletes, commits, rollbacks, and database schema changes are all logged. The transaction log is also called the **forward log** or the **redo log**.

The transaction log is a key component of backup and recovery, and is also essential for data synchronization using MobiLink or for data replication using SQL Remote.

By default, all databases use transaction logs. Using a transaction log is optional, but you should always use a transaction log unless you have a specific reason not to. Running a database with a transaction log provides greater protection against failure, better performance, and the ability to replicate data.

It is recommended that you store the database files and the transaction log on separate disks on the computer. If the dbspace(s) and the transaction log are on the same disk, and a disk failure occurs, everything is lost. However, if the database and transaction log are stored on different disks, then most, if not all, the data can be recovered in the event of a disk failure because you have the full database or the transaction log (from which the database can be recovered).

See "Protecting against media failure" on page 917.

The timestamp of a database or transaction log file is updated only when the file grows or when it is closed. If database operations cause the transaction log file to grow without the database file growing, the timestamp of the transaction log file is more recent than the timestamp of the database file. If the database is shut down, the transaction log file and the database timestamps are updated.

Caution
The database file and the transaction log file must be located on the same physical computer as the database server or accessed via a SAN or iSCSI configuration. Database files and transaction log files located on a remote network directory can lead to poor performance, data corruption, and server instability. See http://www.sybase.com/detail?id=1034790.

When changes are forced to disk

Like the database file, the transaction log is organized into **pages**: fixed size areas of memory. When a change is recorded in the transaction log, it is made to a page in memory. The change is forced to disk when the earlier of the following operations happens:

- The page is full.
- A COMMIT is executed.

Completed transactions are guaranteed to be stored on disk, while performance is improved by avoiding a write to the disk on every operation.

Configuration options are available to allow advanced users to tune the precise behavior of the transaction log. See "cooperative_commits option" on page 526 and "delayed_commits option" on page 534.

See also

- "Controlling transaction log size" on page 24
- "-m dbeng12/dbsrv12 server option" on page 205
- "-m dbeng12/dbsrv12 database option" on page 256
- "sa_disk_free_space system procedure" [*SQL Anywhere Server - SQL Reference*]
- "delete_old_logs option [SQL Remote]" on page 535

Transaction log mirrors

A **transaction log mirror** is an identical copy of the transaction log, maintained at the same time as the transaction log. If a database has a transaction log mirror, every database change is written to both the transaction log and the transaction log mirror. By default, databases do not have transaction log mirrors.

A transaction log mirror provides extra protection for critical data. It enables complete data recovery if a media failure on the transaction log occurs. A transaction log mirror also enables a database server to perform automatic validation of the transaction log on database startup.

It is recommended that you use a transaction log mirror when running high-volume or critical applications. For example, at a consolidated database in a SQL Remote setup, replication relies on the transaction log, and if the transaction log is damaged or becomes corrupt, data replication can fail.

If you are using a transaction log mirror, and an error occurs while trying to write to one of the logs (for example, if the disk is full), the database server stops. The purpose of a transaction log mirror is to ensure complete recoverability if a media failure occurs on either log device; this purpose would be lost if the server continued with a single transaction log.

You can specify the -fc option when starting the database server to implement a callback function when the database server encounters a file system full condition. See "-fc dbeng12/dbsrv12 server option" on page 181.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Where to store the transaction log mirror

There is a performance penalty for using a transaction log mirror because each database log write operation must be performed twice. The performance penalty depends on the nature and volume of database traffic and on the physical configuration of the database and logs.

A transaction log mirror should be kept on a separate device from the transaction log. This improves performance, and if either device fails, the other copy of the log keeps the data safe for recovery.

Alternatives to a transaction log mirror

Alternatives to a transaction log mirror are to use the following configurations:

- database mirroring. See "Introduction to database mirroring" on page 945.
- a disk controller that provides hardware mirroring. Generally, hardware mirroring is more expensive than operating-system level software mirroring, but it provides better performance.
- operating-system level software mirroring, as provided by Microsoft Windows.

Live backups provide additional protection with some similarities to using a transaction log mirror. See "Differences between live backups and transaction log mirrors" on page 892.

For information about creating a database with a transaction log mirror, see "Initialization utility (dbinit)" on page 799.

For information about changing an existing database to use a transaction log mirror, see "Transaction Log utility (dblog)" on page 862.

Change the location of a transaction log

The database cannot be running when you change the location of the transaction log.

For more information about how to choose the location of a transaction log, see "The transaction log" on page 21.

To change the location of a transaction log (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database.
2. From the **Tools** menu, choose **SQL Anywhere 12** » **Change Log File Settings**.
3. Follow the instructions in the **Change Log File Settings Wizard**.

To change the location of a transaction log mirror for an existing database (command line)

1. Ensure that the database is not running.
2. Run the following command:

```
dblog -t new-transaction-log-file database-file
```

See also

- "Transaction Log utility (dblog)" on page 862

Start a transaction log mirror for an existing database

Using the Transaction Log utility, you can maintain the transaction log mirror for an existing database any time the database is not running.

To start a transaction log mirror for an existing database (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database.
2. From the **Tools** menu, choose **SQL Anywhere 12** » **Change Log File Settings**.
3. Follow the instructions in the **Change Log File Settings Wizard**.

To start a transaction log mirror for an existing database (command line)

1. Ensure that the database is not running.
2. Run the following command:

```
dblog -m mirror-file database-file
```

You can also use the dblog utility and Sybase Central to stop a database from using a transaction log mirror.

See also

- "Transaction Log utility (dblog)" on page 862

Controlling transaction log size

The size of the transaction log can also affect recovery times. You can control transaction log file growth by ensuring that all your tables have compact primary keys. If you perform updates or deletes on tables that do not have a primary key or a unique index not allowing NULL, the entire contents of the affected rows are entered in the transaction log. If a primary key is defined, the database server needs to store only the primary key column values to uniquely identify a row. If the table contains many columns or wide columns, the transaction log pages fill up much faster if no primary key is defined. In addition to taking up disk space, this extra writing of data affects performance.

If a primary key does not exist, the server looks for a UNIQUE NOT NULL index on the table (or a UNIQUE constraint). A UNIQUE index that allows NULL is not enough.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "-m dbeng12/dbsrv12 server option" on page 205
- "-m dbeng12/dbsrv12 database option" on page 256
- "sa_disk_free_space system procedure" [*SQL Anywhere Server - SQL Reference*]
- "delete_old_logs option [SQL Remote]" on page 535

Determine which connection has an outstanding transaction

If you are performing a backup that renames or deletes the transaction log, incomplete transactions are carried forward to the new transaction log.

You can use a system procedure to determine which user has outstanding transactions. If there are not too many connections, you can also use the SQL Anywhere Console utility to determine which connection has outstanding transactions. If necessary, you can disconnect the user with a DROP CONNECTION statement.

To determine which connection has an outstanding transaction (SQL)

1. Connect to the database from Interactive SQL.
2. Execute the sa_conn_info system procedure:

   ```
   CALL sa_conn_info;
   ```

3. Inspect the **UncommitOps** column to see which connection has uncommitted operations.

   See "sa_conn_info system procedure" [*SQL Anywhere Server - SQL Reference*].

To determine which connection has an outstanding transaction (SQL Anywhere Console utility)

1. Connect to the database from the SQL Anywhere Console utility.

   For example, the following command connects to the default database using the user ID DBA and password sql:

   ```
   dbconsole -c "UID=DBA;PWD=sql"
   ```

   See "SQL Anywhere Console utility (dbconsole)" on page 848.

2. Double-click each connection, and inspect the **Uncommitted Ops** entry to see which users have uncommitted operations.

See also

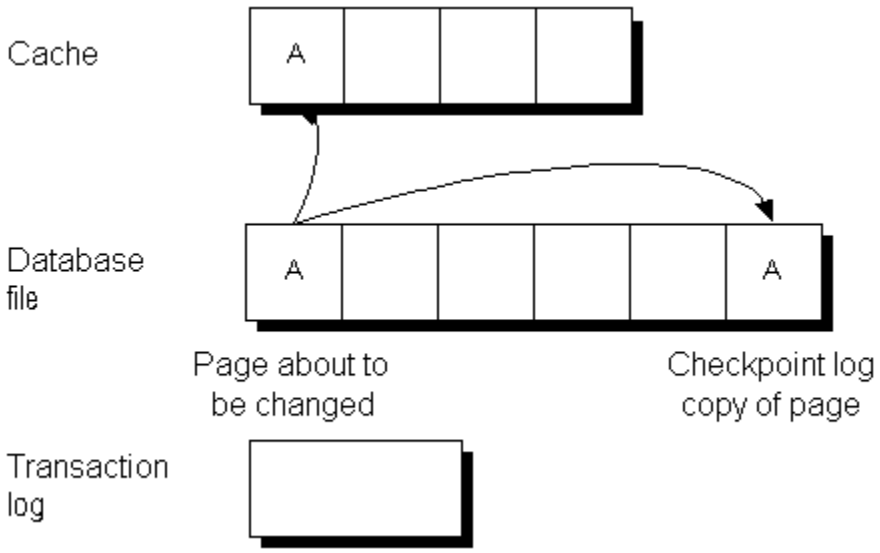
- "DROP CONNECTION statement" [*SQL Anywhere Server - SQL Reference*]

Understanding the checkpoint log

The database file is composed of pages: fixed size portions of hard disk. The **checkpoint log** is located at the end of the database file and is stored in the system dbspace. Pages are added to the checkpoint log as necessary during a session, and at the end of the session, a history of the checkpoint log usage is stored in the database. This history is used to determine an appropriate size for the checkpoint log in future sessions.

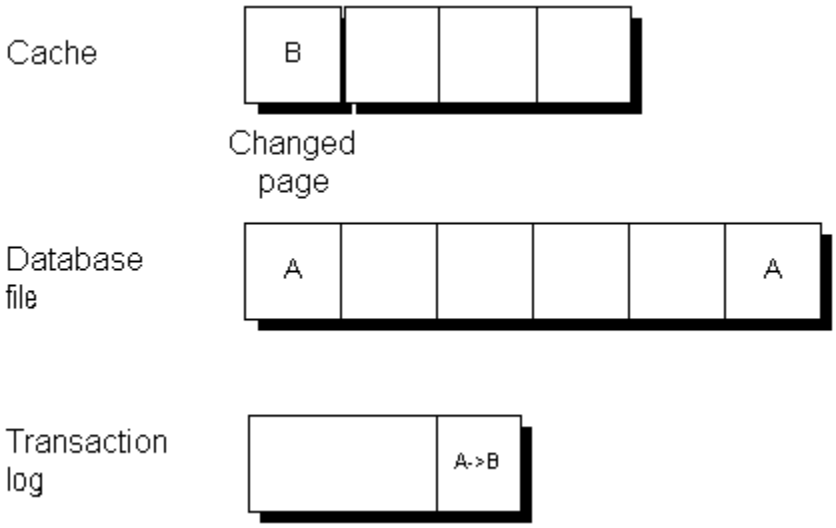
Before any page is updated (made **dirty**), the database server performs the following operations:

- It reads the page into memory, where it is held in the database cache.
- It makes a copy of the original page. These copied pages are the checkpoint log.

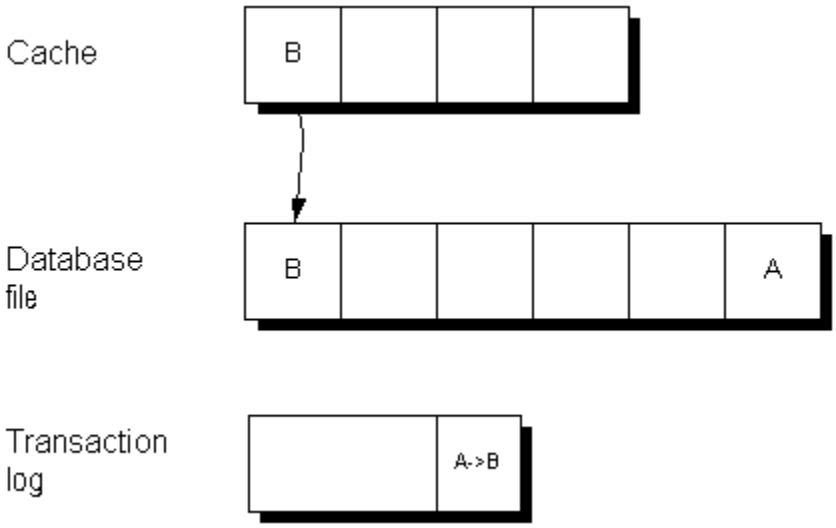


Changes made to the page are applied to the copy in the cache. For performance reasons they are not written immediately to the database file on disk.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0



When the cache is full, the changed page may get written out to disk. The copy in the checkpoint log remains unchanged.



Understanding checkpoints

A **checkpoint** is a point at which all dirty pages are written to disk and therefore represents a known consistent state of the database on disk. Following a checkpoint, the contents of the checkpoint log are deleted. The empty checkpoint log pages remain in the checkpoint log within a given session and can be reused for new checkpoint log data. As the checkpoint log increases in size, so does the database file.

At a checkpoint, all the data in the database is held on disk in the database file. The information in the database file matches that in the transaction log. During recovery, the database is first recovered to the most recent checkpoint, and then changes since that checkpoint are applied.

At the end of each session, a history of the checkpoint log usage is maintained in the database and is used to determine an appropriate size for the checkpoint log for the next session.

The database server can initiate a checkpoint and perform other operations while the checkpoint takes place. However, if a checkpoint is already in progress, then any operation such as an ALTER TABLE or CREATE INDEX statement that initiates a new checkpoint must wait for the current checkpoint to finish.

See also

- "Backup and recovery restrictions" on page 893
- "Understanding backups" on page 923
- "How the database server decides when to checkpoint" on page 924

Using the utility database

The **utility database** is a phantom database with no physical representation. This feature allows you to execute database file administration statements such as CREATE DATABASE without first connecting to an existing physical database. The utility database has no database file, and therefore it cannot contain data.

The utility database is named **utility_db**. If you attempt to create or start a database with this name, the operation fails.

Executing the following statement after connecting to the utility database creates a database named *new.db* in the directory *c:\temp*.

```
CREATE DATABASE 'c:\\temp\\new.db';
```

See "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

You can also retrieve values of connection properties and server properties using the utility database.

For example, executing the following statement against the utility database returns the default collation sequence, which will be used when creating a database:

```
SELECT PROPERTY( 'DefaultCollation' );
```

For information about connection and database server properties, see:

- "Connection properties" on page 619
- "Database server properties" on page 644

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Allowed statements for the utility database

Only the following statements can be executed when connected to the utility database:

- ALTER DATABASE *dbfile* ALTER TRANSACTION LOG (see "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*])
- "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE DECRYPTED DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE DECRYPTED FILE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE ENCRYPTED DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE ENCRYPTED FILE statement" [*SQL Anywhere Server - SQL Reference*]
- "DROP DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- CREATE USER DBA IDENTIFIED BY *new-password* (see "CREATE USER statement" [*SQL Anywhere Server - SQL Reference*])
- "RESTORE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- REVOKE CONNECT FROM DBA (see "REVOKE statement" [*SQL Anywhere Server - SQL Reference*])
- SET TEMPORARY OPTION progress_messages = [OFF | RAW | FORMATTED] (see "SET OPTION statement" [*SQL Anywhere Server - SQL Reference*])
- SELECT statement without a FROM or WHERE clause (see "SELECT statement" [*SQL Anywhere Server - SQL Reference*])
- "START DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "STOP DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "STOP SERVER statement" [*SQL Anywhere Server - SQL Reference*]

See also

- "Specifying the permissions required to execute file administration statements" on page 478

Connecting to the utility database

You can start the utility database on a database server by specifying utility_db as the database name when connecting to the server. You can use the -su server option to set the utility database password for the DBA user, or to disable connections to the utility database. If the -su option is not specified when starting the utility database, then the user ID and password requirements are different for the personal server and the network server.

For the personal database server, if -su is not specified, then there are no security restrictions for connecting to the utility database. For the personal server, you must specify the user ID DBA. You must also specify a password, but it can be any password. It is assumed that anybody who can connect to the personal database server can access the file system directly so no attempt is made to screen users based on passwords.

To avoid typing the utility database password in plain text, when using the -su option, you can create a file that contains the password and then obfuscate it using the dbfhide utility. For example, suppose the file named *util_db_pwd.cfg* contains the utility database password. You could obfuscate this file using dbfhide and rename it to *util_db_pwd_hide.cfg*:

```
dbfhide util_db_pwd.cfg util_db_pwd_hide.cfg
```

The *util_db_pwd_hide.cfg* file can then be used to specify the utility database password:

```
dbsrv12 -su @util_db_pwd_hide.cfg -n my_server c:\mydb.db
```

See "File Hiding utility (dbfhide)" on page 794.

For the network server, if -su is not specified, then you must specify the user ID DBA, and the password that is held in the *util_db.ini* file, stored in the same directory as the database server executable file. As this directory is on the server, you can control access to the file, and thereby control who has access to the password. The password is case sensitive.

Note
The *util_db.ini* file is deprecated. You should use the -su server option to specify the password for the utility database's DBA user. See "-su dbeng12/dbsrv12 server option" on page 225.

To connect to the utility database on the personal server (Interactive SQL)

1. Start a database server with the following command:

   ```
   dbeng12 -n TestEng
   ```

   For additional security, the -su option can be used to specify the utility database password.

2. Start Interactive SQL.

3. In the **Connect** window, specify the following information.
   a. In the **User ID** field, type **DBA**.
   b. In the **Password** field, type any non-blank password. The password itself is not checked, but the field must not be empty.
   c. From the **Action** dropdown list, choose **Connect To A Running Database On Another Computer**.

4. In the **Database Name** field, type **utility_db**.

5. In the **Server Name** field, type **TestEng**.

6. Click **Connect**.

   Interactive SQL connects to the utility database on the personal server named TestEng.

To connect to the utility database on the network server (Interactive SQL)

1. Start a database server with the following command:

   ```
   dbsrv12 -n TestEng -su 9Bx231K
   ```

2. Start Interactive SQL.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

3. In the **Connect** window, type **DBA** for the **User ID**, and type the password specified by the -su option. In this example, the password is **9Bx231K**.
4. From the **Action** dropdown list, choose **Connect To A Database Running On Another Computer**.
5. In the **Database Name** field, type **utility_db**.
6. In the **Server Name** field, type **TestEng**.
7. Click **Connect**.

   Interactive SQL connects to the utility database on the network server named TestEng.

See "SQL Anywhere database connections" on page 86 and "-su dbeng12/dbsrv12 server option" on page 225.

Note
When you are connected to the utility database, executing REVOKE CONNECT FROM DBA disables future connections to the utility database. This means that no future connections can be made to the utility database unless you use a connection that existed before the REVOKE CONNECT was done, or restart the database server. See "REVOKE statement" [*SQL Anywhere Server - SQL Reference*].

Using util_db.ini with network database servers (deprecated)

Using *util_db.ini* relies on the physical security of the computer hosting the database server since the *util_db.ini* file can be easily read using a text editor.

For the network server, by default you cannot connect to the utility database without specifying -su or using *util_db.ini*. If you use *util_db.ini*, the file holds the password and is located in the same directory as the database server executable and contains the text:

```
[UTILITY_DB]
PWD=password
```

To protect the contents of the *util_db.ini* file from casual direct access, you can add simple encryption to the file using the File Hiding utility (dbfhide). You can also use operating system features to limit access to the server file system.

For more information about obfuscating *.ini* files, see "Hiding the contents of .ini files" on page 397.

Erasing a database

Erasing a database deletes all tables and data from disk, including the transaction log that records alterations to the database. All database files are read-only to prevent accidental modification or deletion of database files. By default, you need DBA authority to erase a database. You can change the required permissions by using the database server -gu option. See "-gu dbeng12/dbsrv12 server option" on page 198.

In Sybase Central, you can erase a database using the **Erase Database Wizard**.

In Interactive SQL, you can erase a database using the DROP DATABASE statement.

You can also erase a database from a command line with the dberase utility. However, the dberase utility does not erase dbspaces. If you want to erase a dbspace, you can do so with the DROP DATABASE statement or using the **Erase Database Wizard** in Sybase Central.

The database to be erased must not be running when the dberase utility, the **Erase Database Wizard**, or DROP DATABASE statement is used. You must be connected to a database to drop another database.

For information about connecting to the utility database, see "Connecting to the utility database" on page 29.

Windows Mobile databases must be erased manually. See "Erase a Windows Mobile database" on page 362.

To erase a database (Sybase Central)

1. Choose **Tools** » **SQL Anywhere 12** » **Erase Database**.
2. Follow the instructions in the wizard.

Tip
You can also access the **Erase Database Wizard** from within Sybase Central by using any of the following methods:

- Selecting a database server, and choosing **File** » **Erase Database**.
- Right-clicking a server, and choosing **Erase Database**.

To erase a database (SQL)

1. Connect to a database other than the one you want to erase. For example, connect to the utility database.
2. Execute a DROP DATABASE statement.

   For example, the following DROP DATABASE statement erases a database named temp.

   ```
   DROP DATABASE 'c:\\temp\\temp.db';
   ```

See "DROP DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

To erase a database (command line)

- Run the dberase utility.

  For example, the following command removes the temp database.

  ```
  dberase c:\temp\temp.db
  ```

See "Erase utility (dberase)" on page 793.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Using SQL Anywhere database servers

SQL Anywhere provides two versions of the database server:

- **The personal database server** This executable does not support client/server communications across a network. Although the personal database server is provided for single-user, same-computer use —for example, as an embedded database server—it is also useful for development work.

  On Windows operating systems, except Windows Mobile, the name of the personal server executable is *dbeng12.exe*. On Unix operating systems its name is dbeng12. Only the network server is supported on Windows Mobile.

- **The network database server** This executable supports client/server communications across a network, and is intended for multi-user use.

  On Windows operating systems, including Windows Mobile, the name of the network server executable is *dbsrv12.exe*. On Linux and Unix operating systems, the name is dbsrv12.

Server differences

The request-processing engine is identical in both the personal and network servers. Each one supports exactly the same SQL, but there are some database features that are only supported by the network database server, such as database mirroring and in-memory mode. A database created with a personal database server can be used with a network database server and vice versa. The main differences include:

- **Network protocol support** Only the network server supports communications across a network. See "Selecting communications protocols" on page 76.

- **Number of connections** The personal server has a limit of ten simultaneous connections. The limit for the network server depends on your license. See "Server Licensing utility (dblic)" on page 833.

- **Number of CPUs** With per-seat licensing, the network database server uses all CPUs available on the computer (the default). With CPU-based licensing, the network database server uses only the number of processors you are licensed for. The number of CPUs that the network database server can use may also be affected by your SQL Anywhere edition or the -gt server option. The personal database server is limited to a single processor. See:

  - "Editions and licensing" [*SQL Anywhere 12 - Introduction*]
  - "-gt dbeng12/dbsrv12 server option" on page 195

- **Startup defaults** To reflect their use as a personal server and a network server for many users, the startup defaults are slightly different for each.

Network software requirements

If you are running a SQL Anywhere network server, you must have appropriate networking software installed and running.

The SQL Anywhere network server is available for Windows, Linux, and Unix operating systems.

SQL Anywhere supports the TCP/IP network protocol.

See also

- "The SQL Anywhere database server" on page 147

Start the database server

The way you start the database server varies slightly depending on the operating system you use. As well, you can specify commands in several ways, depending on your operating system:

- Run the command at a command prompt.
- Place the command in a shortcut or desktop icon.
- Run the command in a batch file.
- Include the command as a StartLine (START) connection parameter in a connection string. See "StartLine (START) connection parameter" on page 308.

There are slight variations in how you specify the basic command from platform to platform.

Notes

- Except where otherwise noted, these commands start the personal server (**dbeng12**). To start a network server, replace **dbeng12** with **dbsrv12**.
- If the database file is in the starting directory for the command, you do not need to specify *path*.
- If you do not specify a file extension in *database-file*, the extension *.db* is assumed.

Examples of starting a database server

Command	Comments	More information
On Windows, from the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **SQL Anywhere** » **Personal Server**	The **Server Startup Options** window appears where you can specify information to start a personal database server (dbeng12 executable). You can also start a database on the database server.	
On Windows, from the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **SQL Anywhere** » **Network Server**	The **Server Startup Options** window appears where you can specify information to start a personal database server (dbsrv12 executable). You can also start a database on the database server.	

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Command	Comments	More information
`dbeng12 demo`	Use a database file name in a connection string. Execute this command in the directory where *demo.db* is located to start both a personal server and a database called *demo.db*:	"Connecting to an embedded database" on page 131
`dbeng12 path/database-file`	Unix	
`dbeng12 path\database-file`	This command starts the database server on Windows Mobile. If you omit the database file, the **Server Startup Options** window appears where you can locate a database file by clicking **Browse**.	"Connecting to a database running on a Windows Mobile device" on page 348

See also

- "The SQL Anywhere database server" on page 147
- "Network protocol options" on page 311
- "Using SQL command files" [*SQL Anywhere Server - SQL Usage*]
- "Troubleshooting database server startup" on page 998

Stopping the database server

You can stop the database server by:

- Clicking **Shut Down** on the database server messages window.
- Using the dbstop utility.

  The dbstop utility is useful in batch files, or for stopping a server on another computer. It requires a connection string in its command. See "Stop Server utility (dbstop)" on page 851.
- Letting it shut down automatically by default when the application disconnects.
- Pressing Q when the database server messages window has the focus on Unix.

Example

To stop a database server using the dbstop utility

1. Start a database server. For example, the following command executed from the SQL Anywhere installation directory starts a server named Ottawa using the sample database:

   ```
   dbsrv12 -n Ottawa samples-dir\demo.db
   ```

2. Stop the server using dbstop:

```
dbstop -c "Server=Ottawa;UID=DBA;PWD=sql"
```

Who can stop a database server?

When you start a database server, you can use the -gk option to set the level of permissions required for users to stop the server with dbstop. For personal database servers, the default is all. The default level of permissions required is DBA for network database servers, but you can also set the value to all or none. (However, anyone at the computer can click **Shut Down** on the database server messages window.)

See also

- "-gk dbeng12/dbsrv12 server option" on page 187

Shutting down operating system sessions

If you close an operating system session where a database server is running, or if you use an operating system command to stop the database server, the server shuts down, but not cleanly. The next time the database loads, recovery is required, and happens automatically.

For more information about recovery, see "Backup and data recovery" on page 887.

It is better to stop the database server explicitly before closing the operating system session.

Examples of commands that do not stop a server cleanly include:

- Stopping the process in the Windows Task Manager
- Using a Unix slay or kill command

Starting and stopping databases

A database server can have more than one database loaded at a time. You can start databases and start the database server at the same time, as follows:

```
dbeng12 demo sample
```

Caution
The database file must be on the same computer as the database server. Managing a database file that is located on a network drive can lead to file corruption.

Starting a database on a running server

You can also start databases after starting a server in one of the following ways:

- Connect to a database using a DatabaseFile (DBF) connection parameter while connected to a server. The DatabaseFile (DBF) connection parameter specifies a database file for a new connection. The database file is started on the current server.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See "Connecting to an embedded database" on page 131, or "DatabaseFile (DBF) connection parameter" on page 280.

- Use the START DATABASE statement, or choose **Start Database** from the **File** menu in Sybase Central when you have a server selected.

  See "START DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

Limitations

- The server holds database information in memory using pages of a fixed size. Once a server has been started, you cannot start a database that has a larger page size than the server.

  See "Setting a maximum page size" on page 39.

- The -gd server option decides the permissions required to start databases.

Start a database

With both Sybase Central and Interactive SQL, you can start a database without connecting to it.

To start a database on a database server without connecting (Sybase Central)

1. Select the database server and then choose **File** » **Start Database**.
2. In the **Start Database** window, enter the required values.

   The database appears under the database server as a disconnected database.

To start a database on a server without connecting (SQL)

- Execute a START DATABASE statement.

See "START DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

Start the database file *c:\temp\temp.db* on the database server named sample.

```
START DATABASE 'c:\\temp\\temp.db'
AS tempdb ON 'sample'
AUTOSTOP OFF;
```

You must be connected to a database to start another database.

The AUTOSTOP OFF connection parameter prevents the database from being stopped automatically when all connections have been disconnected. It is used here to illustrate a point later on in the discussion.

For more details about starting a database, see "Using SQL Anywhere database servers" on page 33.

Stop a database

You can stop a database by:

- Disconnecting from a database started by a connection string. Unless you explicitly set the AutoStop (ASTOP) connection parameter to NO, this happens automatically.

  See "AutoStop (ASTOP) connection parameter" on page 269.

- Using the STOP DATABASE statement from Interactive SQL or embedded SQL.

  See "STOP DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

With both Sybase Central and Interactive SQL, you can stop a database running on a database server. You cannot stop a database you are currently connected to. You must first disconnect from the database, and then stop it. You must be connected to another database on the same database server to stop a database.

For more details about stopping a database, see "Using SQL Anywhere database servers" on page 33.

To stop a database on a database server after disconnecting (Sybase Central)

1. Make sure you are connected to at least one other database on the same database server. If there is no other database running on the database server, you can connect to the utility database.
2. Select the database you want to stop and choose **File** » **Stop Database**.

When disconnecting from the database, the database may disappear from the left pane. This occurs if your connection was the only remaining connection, and if AutoStop was specified when the database was started. AutoStop causes the database to be stopped automatically when the last connection disconnects.

To stop a database on a server after disconnecting (SQL)

1. If you aren't connected to any database on the server, then connect to a database such as the utility database.
2. Execute a STOP DATABASE statement.

Example

The following statements connect to the utility database and stops the tempdb database.

```
CONNECT to 'TestEng' DATABASE utility_db
AS conn2
USER 'DBA'
IDENTIFIED BY 'sql';
STOP DATABASE tempdb;
```

You must be connected to a database to stop another database.

See also

- "Connecting to the utility database" on page 29

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Configuring database servers

- You can choose from many options to specify such features as how much memory to use as cache, how many CPUs to use (on multi-processor computers running a network database server), and which network protocols to use (network server only). Options are one of the major ways of tuning SQL Anywhere behavior and performance. See "The SQL Anywhere database server" on page 147.

- You can run the server as a Windows service. When you run the server as a service, the server continues running even when you log off the computer. See "Running the database server outside the current session" on page 57.

- You can start the personal server from an application and shut it down when the application has finished with it. This configuration is typical when using the database server as an embedded database. See "Connecting to an embedded database" on page 131.

See also

- "Start the database server" on page 34

Setting a maximum page size

The database server cache is arranged in **pages**—fixed-size areas of memory. Since the database server uses a single cache for its lifetime (until it is shut down), all pages must have the same size.

A database file is also arranged in pages, with a size that is specified when the database is created. Every database page must fit into a cache page. By default, the database server page size is the same as the largest page size of the databases that are specified when the database server is started. Once the database server starts, you cannot start a database with a larger page size than the database server page size.

To allow databases with larger page sizes to be started after startup, you can force the database server to start with a specific page size by using the -gp option. If you use larger page sizes, remember to increase your cache size. A cache of the same size accommodates only a fraction of the number of the larger pages, leaving less flexibility in arranging the space.

The following command starts a database server that reserves a 64 MB cache and can accommodate databases of page sizes up to 8192 bytes.

```
dbsrv12 -gp 8192 -c 64M -n myserver
```

See also

- "-gp dbeng12/dbsrv12 server option" on page 193

Naming the database server and the databases

You can use -n as a database server option (to name the database server) or as a database option (to name the database).

Client applications can use connection parameters that specify the database server name and database name when connecting to a database. The database server name appears on the desktop icon and in the title bar of the database server messages window.

You cannot create a database or start a database server with the name utility_db because this name is reserved for the SQL Anywhere utility database. See "Using the utility database" on page 28.

Naming the database server

Providing a database server name helps avoid conflicts with other database server names on your network. It also provides a meaningful name for users of client applications. The database server keeps its name for its lifetime (until it is shut down). If you don't provide a database server name, the database server is given the name of the first database started.

You can name the database server by supplying a -n option before the first database file. For example, the following command starts a database server running the sample database and gives the database server the name Cambridge:

```
dbeng12 -n Cambridge samples-dir\demo.db
```

If you supply a database server name, you can start a database server without starting a database. The following command starts a database server named Galt with no database started:

```
dbeng12 -n Galt
```

The maximum length of the database server name is 250 bytes.

For more information about starting databases on a running server, see "Starting and stopping databases" on page 36.

Note
On Windows and Unix, version 9.0.2 and earlier clients cannot connect to version 10.0.0 and later database servers with names longer than the following lengths:

- 40 bytes for Windows shared memory
- 31 bytes for Unix shared memory
- 40 bytes for TCP/IP

Naming databases

You may want to provide a meaningful database name for users of client applications. The database is identified by that name until it is stopped. The maximum length for database names is 250 bytes.

If you don't provide a database name, the default name is the root of the database file name (the file name without the *.db* extension). For example, in the following command the first database is named mydata, and the second is named mysales.

```
dbeng12 c:\mydata.db c:\sales\mysales.db
```

You can name databases by supplying a -n option following the database file. For example, the following command starts the sample database and names it MyDB:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
dbeng12 samples-dir\demo.db -n MyDB
```

Case sensitivity

Database server names and database names are case insensitive as long as the character set is single-byte. See "Connection strings and character sets" on page 410.

Using configuration files to store database server startup options

You can store the set of options used to start a database server in a configuration file and invoke that file in a database server command. The configuration file can contain options on several lines. For example, the following configuration file starts a personal database server and the sample database. It sets a cache of 10 MB, and names this instance of the personal server **Elora**. Lines with # as the first character in the line are treated as comments.

```
# Configuration file for server Elora
-n Elora
-c 10M
samples-dir\demo.db
```

In the example, *samples-dir* is the name of your SQL Anywhere samples directory. On Unix, use a forward slash instead of the backslash in the file path.

For information about *samples-dir*, see "Samples directory" on page 392.

If you name the file containing these options*sample.cfg*, you could use the file as follows:

```
dbeng12 @sample.cfg
```

See also

- "@data dbeng12/dbsrv12 server option" on page 157
- "Using configuration files" on page 763
- "Using conditional parsing in configuration files" on page 765

Logging database server actions

The **database server message log** contains informational messages, errors, warnings, and messages from the MESSAGE statement. These messages can appear in the following locations:

- the database server messages window (a system tray icon on Windows)
- the Sybase Central **Server Messages And Executed SQL** pane
- the SQL Anywhere Console utility
- the database server message log file
- a command prompt window or shell when running the database server as a command line application
- the Unix Syslog

See also

- "-o dbeng12/dbsrv12 server option" on page 208
- "-oe dbeng12/dbsrv12 server option" on page 209
- "-on dbeng12/dbsrv12 server option" on page 209
- "-os dbeng12/dbsrv12 server option" on page 210
- "-ot dbeng12/dbsrv12 server option" on page 211

Log database server messages to a file

By default, database server messages are sent to the database server messages window. You can send the output to a log file using the -o option. The following command sends output to a log file named *mydbserver_messages.txt*:

```
dbsrv12 -o mydbserver_messages.txt -c ...
```

You can control the size of the database server message log file, and specify what you want done when a file reaches its maximum size:

- Use the -o option to specify that a database server message log file should be used and to provide a name.
- Use the -ot option to specify that a database server message log file should be used and provide a name when you want the previous contents of the file to be deleted before new messages are sent to it.
- In addition to -o or -ot, use the -on option to specify the size at which the database server message log file is renamed with the extension *.old* and a new file is started with the original name.
- In addition to -o or -ot, use the -os option to specify the size at which a new database server message log file is started with a new name based on the date and a sequential number.

You can specify a separate file where startup errors, fatal errors, and assertions are logged using the -oe option.

It is recommended that you do not end the database server message log file name with *.log* because this can create problems for utilities that perform operations using the transaction log.

See also

- "-o dbeng12/dbsrv12 server option" on page 208
- "-oe dbeng12/dbsrv12 server option" on page 209
- "-on dbeng12/dbsrv12 server option" on page 209
- "-os dbeng12/dbsrv12 server option" on page 210
- "-ot dbeng12/dbsrv12 server option" on page 211

Log SQL statements in Sybase Central

As you work with a database in Sybase Central, the application automatically generates SQL statements depending on your actions. You can keep track of these statements in a separate pane, called **Server Messages And Executed SQL**, or save the information to a file. The **Server Messages And Executed**

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

SQL pane has a tab for each database and database server. The tab for database servers contains the same information as the database server messages window.

When you work with Interactive SQL, you can also log statements that you execute. See "Logging commands" on page 713.

To log SQL statements generated by Sybase Central to the Server Messages and Executed SQL pane

1. Use the **SQL Anywhere 12** plug-in to connect to the database.
2. Choose **View** » **Server Messages And Executed SQL**.
3. In the **Server Messages And Executed SQL** pane, click the tab with the database icon.
4. Right-click and choose **Options**.
5. Edit the logging options.
6. Click **Save**.

To log SQL statements generated by Sybase Central to a file (Sybase Central)

1. Connect to a database.
2. In the left pane, right-click the database and choose **Start Logging Database Changes**.
3. Specify a file name and click **Save**.

To stop logging database changes (Sybase Central)

- In the left pane, right-click the database and choose **Stop Logging Database Changes**.

Suppressing Windows event log messages

You can suppress Windows event log entries by setting a registry entry. The registry entry is *Software\Sybase\SQL Anywhere\12.0*. This entry can be placed in either the HKEY_CURRENT_USER or HKEY_LOCAL_MACHINE hive.

To control event log entries, set the EventLogMask key, which is of type REG_DWORD. The value is a bit mask containing the internal bit values for the different types of event messages:

```
errors      EVENTLOG_ERROR_TYPE        0x0001
warnings    EVENTLOG_WARNING_TYPE      0x0002
information EVENTLOG_INFORMATION_TYPE  0x0004
```

For example, if the EventLogMask key is set to 0, no messages appear. When you set this key to 1, informational and warning messages do not appear, but errors do. The default setting (no entry present) is for all message types to appear.

When changing the setting of the EventLogMask key, you must restart the database server for the change to take effect.

See also

- "Network protocol options" on page 311

Controlling performance and memory from the command line

Several settings can affect database server performance, including:

- **Cache size** The amount of cache memory available to the database server can be a key factor in affecting performance. Generally speaking, the more memory made available to the database server, the faster it performs. The cache holds information that may be required more than once. Accessing information in cache is faster than accessing it from disk. The default initial cache size is computed based on the amount of physical memory, the operating system, and the size of the database files. The database server automatically adjusts the cache size as necessary. See "Dynamic cache sizing" [*SQL Anywhere Server - SQL Usage*].

  The database server messages window displays the size of the cache at startup, and you can use the following statement to obtain the current size of the cache:

  ```
  SELECT PROPERTY( 'CurrentCacheSize' );
  ```

  For more information about performance tuning, see "Improving database performance" [*SQL Anywhere Server - SQL Usage*].

  The following table summarizes the database server options available for controlling the cache.

Cache feature	Database server option	Used for	See
Cache size	-c	Sets the initial amount of memory for the database server cache	"-c dbeng12/dbsrv12 server option" on page 159
	-ca 0	Enforces a static cache size	"-ca dbeng12/dbsrv12 server option" on page 161

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Cache feature	Database server option	Used for	See
	-ch	Sets the maximum cache size for automatic cache resizing	“-ch dbeng12/dbsrv12 server option” on page 163
	-chx	Sets the maximum cache size for automatic cache resizing without reserving address space for non-cache use (32-bit database servers only)	“-chx dbeng12/dbsrv12 server option” on page 164
	-cl	Sets the minimum cache size for automatic cache resizing	“-cl dbeng12/dbsrv12 server option” on page 166
	-cs	Displays statistics about dynamic cache size changes in the database server messages window	“-cs dbeng12/dbsrv12 server option” on page 169

Cache feature	Database server option	Used for	See
Cache warming	-cc	Collects information about database pages that can be used for cache warming the next time the database is started	"-cc dbeng12/dbsrv12 server option" on page 162
	-cr	Warms the cache with database pages	"-cr dbeng12/dbsrv12 server option" on page 169
	-cv	Displays messages about cache warming in the database server messages window	"-cv dbeng12/dbsrv12 server option" on page 170
Address Windowing Extensions (AWE) cache (deprecated)	-cm	Sets the amount of address space allocated for an AWE cache on Windows	"-cm dbeng12/dbsrv12 server option" on page 167
	-cw	Enables the use of AWE on Windows	"-cw dbeng12/dbsrv12 server option (deprecated)" on page 171

Note
The use of AWE is deprecated. It is recommended that you use the 64-bit version of the SQL Anywhere database server on a 64-bit Windows operating system if you require a large cache.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Multiprogramming level** The database server's multiprogramming level specifies the maximum number of database server tasks that can execute concurrently. In general, a higher multiprogramming level increases the overall throughput of the database server by permitting more requests to execute simultaneously. However, if the requests compete for the same resources, increasing the multiprogramming level can lead to additional contention and lengthen transaction response time.

  By default, SQL Anywhere automatically adjusts the database server's multiprogramming level. In some cases you can lower the throughput of the system by increasing the multiprogramming level. The following options allow you to control the database server's multiprogramming level manually:

Database server option	sa_server_option value	Description
"-gn dbsrv12 server option" on page 189	CurrentMultiProgrammingLevel	Sets the multiprogramming level of the database server.
"-gna dbsrv12 server option" on page 190	AutoMultiProgrammingLevel	Turns on and off dynamic tuning of the database server's multiprogramming level.
"-gnh dbsrv12 server option" on page 190	MaxMultiprogrammingLevel	Sets the maximum number of tasks that the database server can execute concurrently.
"-gnl dbsrv12 server option" on page 192	MinMultiProgrammingLevel	Sets the minimum number of tasks that the database server can execute concurrently.
"-gns dbsrv12 server option" on page 192	AutoMultiProgrammingLevelStatistics	Controls whether statistics about the automatic changes to the multiprogramming level appear in the database server message log. You can also control this behavior by using the AutoMultiProgrammingLevelStatistics property with the sa_server_option system procedure. See and "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].

  For more information about the multiprogramming level in SQL Anywhere, see "Configuring the database server's multiprogramming level" on page 52.

- **Number of processors** If you are running on a multi-processor computer using a network database server, you can set the number of processors with the -gt option. See "-gt dbeng12/dbsrv12 server option" on page 195 and "Threading in SQL Anywhere" on page 49.

The number of CPUs that the database server can use may also be affected by your license or SQL Anywhere edition. See "Editions and licensing" [*SQL Anywhere 12 - Introduction*].

- **Other performance-related options** There are several options available for tuning network performance, including -gb (database process priority), and -u (buffered disk I/O). See "The SQL Anywhere database server" on page 147.

Controlling permissions from the command line

Some options control the permissions required to perform certain global operations, including permissions to start and stop databases, load and unload data, and create and delete database files. See "Running the database server in a secure fashion" on page 1129.

Running in special modes

You can run SQL Anywhere in special modes:.

- **Read-only** You can run databases in read-only mode by supplying the -r option. Databases that have auditing turned on cannot be started in read-only mode. See "-r dbeng12/dbsrv12 server option" on page 217, and "-r dbeng12/dbsrv12 database option" on page 258.

- **In-memory mode** You can run databases entirely in memory by specifying the -im option. When you run in checkpoint mode only (-im c), the database server does not use a transaction log, but the database can be recovered to the most recent checkpoint. When you run the database in never write mode (-im nw), committed transactions are not written to the database file on disk, and all changes are lost when you shut down the database. Using either in-memory mode, your application can still make changes to the database or access it while the database is active. See "-im dbeng12/dbsrv12 server option" on page 199.

  Separately licensed component required
  In-memory mode requires a separate license. See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

- **Bulk load** This is useful when loading large quantities of data into a database using the Interactive SQL INPUT command. Do not use the -b option if you are using LOAD TABLE to bulk load data. See "-b dbeng12/dbsrv12 server option" on page 158, and "Importing and exporting data" [*SQL Anywhere Server - SQL Usage*].

- **Starting without a transaction log** Use the -f database option for recovery—either to force the database server to start after the transaction log has been lost, or to force the database server to start using a transaction log it would otherwise not find. Note that -f is a database option, not a server option.

  Once the recovery is complete, you should stop your server and restart without the -f option. See "-f dbeng12/dbsrv12 server recovery option" on page 180.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Operating quietly** The database server supports quiet mode. You determine how quiet you want the server to operate, ranging from suppressing messages or the icon in the system tray, to complete silence. To operate a completely silent database server on Windows, specify the -qi, -qs, and -qw options. With these options set, there is no visual indication that the server is running as all icons and all possible startup error messages are suppressed. If you run the database server in quiet mode, you can use either (or both) the -o or -oe options to diagnose errors.

  Note that the -qi and -qs options do not suppress windows caused by the -v (version) and -ep (prompt for database encryption password) server options. See:

  - "-qi dbeng12/dbsrv12 server option" on page 214
  - "-qn dbeng12/dbsrv12 server option" on page 214
  - "-qs dbeng12/dbsrv12 server option" on page 216
  - "-qw dbeng12/dbsrv12 server option" on page 216

Threading in SQL Anywhere

To understand the SQL Anywhere threading model, you must understand the basic terminology and concepts of threading and request processing:

- **Request** A request is a unit of work, such as a query or SQL statement, that is sent to the database server over a connection. The lifetime of a request spans the time from when the request is first received by the database server to the time that the last of the results are returned, cursors are closed, or the request is canceled.

- **Task** A task is a unit of activity that is performed within the database server, and is the smallest unit of work that is scheduled by the database server. Within the database server, each user request becomes at least one task, and possibly more if intra-query parallelism is involved. In addition to user requests, the database server can also schedule its own tasks to perform internal tasks, such as running the database cleaner or processing timers. The maximum number of active tasks that can execute concurrently depends on the size of the worker pool within the kernel. If a user request task arrives at the database server and a worker is assigned to it, the ActiveReq server property is incremented by 1. Once the request completes, the ActiveReq server property is decremented by 1. If, however, there were no available workers to execute the user request task, then the task is queued for execution and the UnschReq server property is incremented by 1. When the task is dequeued for execution because a worker is now available, the UnschReq server property is decremented by 1, and the ActiveReq property is incremented by 1. See "UnschReq server property" on page 659 and "ActiveReq server property" on page 645.

- **Worker** A worker is an abstraction of an executing **thread-of-control** within the database server kernel. On Windows and Linux, workers are implemented using lightweight threads called **fibers**, and on other platforms workers are implemented using operating system threads. Workers execute tasks that are assigned to them by the database server kernel. The database server utilizes a variable-sized pool of workers to handle the server's workload. The size of the pool corresponds to the database server's multiprogramming level.

  Personal servers, as well as network servers on Windows Mobile, have a fixed-sized pool. For these servers, the number of workers created at server startup is controlled by the -gn option. For all other

network servers, the database server creates the number of workers specified by the -gnh option. However, only the number of workers specified by the server multiprogramming level are allowed to service tasks. By default the size of the worker pool is dynamically tuned by the kernel in response to changes in the database server's workload, and can fluctuate between the lower and upper bounds specified by the -gnl and -gnh options respectively. See "Configuring the database server's multiprogramming level" on page 52.

Tasks are assigned to workers on a first-in, first-out (FIFO) basis. Each task's priority is set based on the connection that generated that task. Once a task is assigned to a worker for execution, the kernel's scheduler takes the task's priority into account when allocating CPU time to that worker. During task execution, if a task needs to block for some reason during its processing, such as while waiting for a lock or for I/O to complete, the worker remains coupled to that task. When the task completes, only then does the worker become available to execute other tasks.

- **Thread** A thread, or thread of execution, is an operating system construct that permits concurrent execution within a single process. Every operating system process, including the database server, is executed by at least one, and possibly many threads. A thread is scheduled outside the application by the operating system, and ultimately, all of an application's execution is performed by its threads. On Windows and Linux, the database server creates a fixed number of threads: one operating system thread per CPU core, controlled by the -gtc option. On other platforms, the number of operating system threads created is equivalent to the size of the worker pool and is controlled using the same mechanisms that are used to control the worker pool size. See "-gtc dbeng12/dbsrv12 server option" on page 196 and "-gn dbsrv12 server option" on page 189.

  The number of threads is independent of the number of connections to the database, and the database server does not dedicate a thread to a specific connection. Instead, as tasks enter the database server for execution, they are dynamically assigned a thread from a fixed-size pool of server threads. Once a task is assigned to a thread, the thread processes the task until the task completes or is cancelled.

See also

- "Controlling threading behavior" on page 51
- "Configuring the database server's multiprogramming level" on page 52
- "Parallelism during query execution" [*SQL Anywhere Server - SQL Usage*]
- "-gn dbsrv12 server option" on page 189
- "-gnl dbsrv12 server option" on page 192
- "-gnh dbsrv12 server option" on page 190
- "-gtc dbeng12/dbsrv12 server option" on page 196
- "sa_clean_database system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Transaction blocking and deadlock" [*SQL Anywhere Server - SQL Usage*]

Workers on Unix

On Unix, a worker is implemented using an operating system thread. Scheduling of tasks is therefore controlled by the operating system scheduler and the operating system may choose to preempt the execution of a thread at any time. This preemptive scheduling does not affect the processing of tasks in any visible way; when a thread is scheduled to run again, the task is picked up at the point where it left off. However, task priorities that are inherited from a database connection will have no effect on the

execution scheduling of those tasks. Rather, all tasks will be executed at the same priority level because all operating system thread priorities are equivalent.

Workers on Windows and Linux

On Windows and Linux, workers are implemented using lightweight threads known as **fibers**. An implementation utilizing fibers allows workers to be scheduled amongst themselves co-operatively, rather than being scheduled preemptively by the operating system thread scheduler. The result is that a context switch between fibers can be more finely controlled. The database kernel schedules the fibers to maximize processor affinity and tightly control the execution priority of each worker.

Because fibers do not rely on the operating system scheduler, a fiber must explicitly yield control to another fiber when it is waiting for some other activity to complete. For example, if a task that is executing on a fiber needs to block while waiting for an I/O operation to complete, it relinquishes control to another fiber. The thread hosting the original fiber is free to pick up another fiber immediately and begin its execution without a kernel context switch. If a fiber blocks and does not yield control, it blocks the thread that is hosting it and prevents other fibers from running on that thread. If more than one thread is hosting fibers, only the thread that is hosting the waiting fiber is blocked: other threads are still free to run fibers.

On Windows and Linux platforms there are at least as many fibers created as required by the maximum concurrency setting of the database server, as specified by the -gnh option. The database server utilizes a subset of these fibers during server execution, corresponding to the current server multiprogramming level. For each fiber the operating system must reserve address space for its stack. The amount of stack required for each fiber depends on the setting of the -gss server option. See “-gss dbeng12/dbsrv12 server option” on page 195.

The amount of address space required by a network server for execution stacks is proportional to the maximum multiprogramming level. See “-gn dbsrv12 server option” on page 189.

Controlling threading behavior

There are several factors that control threading behavior, each of which are governed by a server option. Not all of these options are supported on every platform.

- **Multiprogramming level (-gn server option)** The -gn option controls the database server's multiprogramming level. This value determines the maximum number of tasks that may be active at one time. Each database request causes the creation of at least one task, and possibly more if intra-query parallelism is involved. Additionally, the server occasionally schedules tasks to perform internal activities. When the number of tasks in the server exceeds the multiprogramming level, outstanding tasks must wait until a currently-running, or active task, completes. See “-gn dbsrv12 server option” on page 189 and “Configuring the database server's multiprogramming level” on page 52.

- **Stack size per internal execution thread (-gss server option)** You can set the stack size per worker in the database server using the -gss option. The -gss option allow you to lower the address

space requirements of each worker within the database server, which may be useful in environments with limited memory. See "-gss dbeng12/dbsrv12 server option" on page 195.

- **Number of processors (-gt server option)** The -gt option controls the number of processors that the database server uses. See "-gt dbeng12/dbsrv12 server option" on page 195.
- **Processor concurrency (-gtc server option)** The -gtc option specifies the number of logical processors (cores) that the database server uses. See "-gtc dbeng12/dbsrv12 server option" on page 196.

Processor use and threading example

The following example explains how the database server selects CPUs based on the settings of -gt and -gtc. For the following examples, assume you have a system with 4 processors, with 2 cores on each processor. The physical processors are identified with letters, and the cores with numbers, so this system has processing units A0, A1, B0, B1, C0, C1, D0, and D1.

Scenario	Network database server settings
A single CPU license or -gt 1 specified	• -gt 1 • -gtc 2 Threads can execute on A0 and A1.
No licensing restrictions on the CPU with -gtc 5 specified	• -gt 4 • -gtc 5 Threads can execute on A0, A1, B0, C0, and D0.
A database server with a 3 CPU license and -gtc 5 specified	• -gt 3 • -gtc 5 Threads can execute on A0, A1, B0, B1, and C0.
No licensing restrictions on the CPU with -gtc 1 specified	• -gt 4 • -gtc 1 Threads can execute only on A0.

Configuring the database server's multiprogramming level

The database server's **multiprogramming level** is the maximum number of tasks that can be active at a time. When a client-side request arrives at the database server, the task created for that request is assigned to a worker, if one is available. A request with a worker assigned to it is called an **active request**. If all available workers are busy, then the request is placed in a special queue called the unscheduled request queue and the request is classified as an Unscheduled Request. Similarly, an active task is one that is currently being serviced by a worker. An active task may be executing an access plan operator, or performing some other function, but may also be blocked, waiting for a resource (such as an I/O

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

operation, or a lock on a row). An unscheduled task is one that is ready to execute, but is waiting for an available worker. The number of active tasks that can execute simultaneously depends on the number of server threads and the number of logical processors in use on the computer.

SQL Anywhere lets DBAs choose between allowing the database server to dynamically tune the multiprogramming level based on server throughput (the default) or configuring the multiprogramming level manually. You can configure the multiprogramming level settings when you start a database server by specifying database server options (-gna, -gnl, and -gnh), or after the database server is running by using the MinMultiprogrammingLevel, MaxMultiprogrammingLevel, and CurrentMultiprogrammingLevel properties with the sa_server_option system procedure.

The following table summarizes the command line and server options that control the database server's multiprogramming level:

Database server option (starting database servers)	sa_server_option value (running database servers)	Description
-gn	CurrentMultiProgrammingLevel	Sets the multiprogramming level of the database server.
-gna	AutoMultiProgrammingLevel	Turns on and off dynamic tuning of the database server's multiprogramming level
-gnh	MaxMultiprogrammingLevel	Sets the maximum number of tasks that the database server can execute concurrently
-gnl	MinMultiProgrammingLevel	Sets the minimum number of tasks that the database server can execute concurrently

Tuning the multiprogramming level

SQL Anywhere network servers support both dynamic and manual tuning of the server's multiprogramming level.

- **Dynamic tuning of the multiprogramming level** SQL Anywhere network servers can automatically monitor the throughput level of the database server and determine how the multiprogramming level should be adjusted in response to the current workload. The database server uses a hill-climbing algorithm, as well as a parabola approximation approach, to decide on the adjustment that needs to be made. If an increase in the multiprogramming level results in an increase in the database server throughput level, then the database server proceeds with the increase. If the increase in the multiprogramming level results in degradation in throughput, then the database server lowers the multiprogramming level. The database server continuously monitors the throughput level

and changes the multiprogramming level to improve server throughput. For workloads that consist of short bursts of a large number of requests followed by long idle periods, it is best to set the minimum multiprogramming level to the maximum expected concurrency level. This configuration ensures that the database server is responsive during the short bursts of requests.

Dynamic tuning of the multiprogramming level is enabled by default (-gna 1). The -gnl and -gnh database server options set the minimum and maximum values for the multiprogramming level that the dynamic tuning algorithm uses. When dynamic tuning is turned on, the database server also attempts to eliminate thread deadlock issues by automatically increasing the multiprogramming level each time a thread deadlock condition arises. The database server continues increasing the multiprogramming level up to the allowable MaxMultiprogrammingLevel value. Once the MaxMultiprogrammingLevel value is reached, the database server starts returning thread deadlock issues to client applications. See "ThreadDeadlocksAvoided server property" on page 659.

- **Manual tuning of the multiprogramming level** You must turn off dynamic tuning of the multiprogramming level before you can adjust it manually. It is recommended that you test your application's workload to analyze the effects of the database server's multiprogramming level on server throughput and request response times. You can use the Windows Performance Monitor or the Performance Statistics utility (dbstats) on Unix to help you analyze database server behavior when testing your application. Performance statistics can be queried using the PROPERTY, DB_PROPERTY, and CONNECTION PROPERTY functions.

  The performance counters related to active and unscheduled requests should be observed during this analysis. See "ActiveReq server property" on page 645 and "UnschReq server property" on page 659.

  If the number of active requests is always less than the value of the -gn database server option, you can consider lowering the multiprogramming level, but you must take into account the effects of intra-query parallelism, which adds additional tasks to the database server's execution queues. If the effect of intra-query parallelism is marginal, lowering the multiprogramming level can be done safely without reducing overall system throughput. However, if the number of total requests (active + unscheduled) is often larger than the value specified by -gn, then an increase in the multiprogramming level may be warranted. You should consider the performance tradeoffs of increasing the multiprogramming level before changing it. See "Raising the multiprogramming level" on page 55.

See also

- "-gn dbsrv12 server option" on page 189
- "-gna dbsrv12 server option" on page 190
- "-gnh dbsrv12 server option" on page 190
- "-gnl dbsrv12 server option" on page 192
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

Understanding the effects of different multiprogramming level settings

It can be difficult to determine the optimal value for the multiprogramming level. For example, if a database application uses Java stored procedures, or if intra-query parallelism is enabled, then the additional server tasks that are created to process these requests may exceed the current multiprogramming level, and execution of these tasks must wait until another request completes. In these

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

cases, raising the multiprogramming level may be appropriate. You can adjust the multiprogramming level of a network database server at any time. However, for personal database servers you cannot change the multiprogramming level after the server has started.

Raising the multiprogramming level

Often, increases to the multiprogramming level correspondingly increase the database server's overall throughput because an increase to the multiprogramming level allows additional tasks to execute concurrently. However, there are tradeoffs in raising the multiprogramming level that should be considered. They include the following:

- **Increased contention** By increasing the number of concurrent tasks, you may increase the probability of contention between active requests. The contention can involve resources such as schema or row locks, or on data structures and/or synchronization primitives internal to the database server. Such a situation may actually decrease server throughput.

- **Additional server overhead** Each active task requires the allocation and maintenance of a worker and additional bookkeeping structures to control its scheduling. In addition, each worker requires the pre-allocation of address space for its execution stack. The size of the stack varies by platform. On Windows operating systems, the allocation of stack space affects the address space of the database server process, but the stack memory is allocated on demand. On Unix platforms, including Linux, the backing memory for the stack is allocated immediately. As a result, setting a higher multiprogramming level increases the server's memory footprint, and reduces the amount of memory available for the cache manager because the amount of available address space is reduced.

- **Thrashing** The database server can reach a state when it uses significant resources simply to manage its execution overhead, rather than doing useful work for a specific request. This state is commonly called thrashing. Thrashing can occur, for example, when too many active tasks are competing for space in the database server cache, but the cache is not large enough to accommodate the working set of database pages used by the set of active tasks. This situation can result in page stealing, in a manner similar to that which can occur with operating systems.

- **Impact on query processing** The database server selects a maximum number of memory-intensive requests that can be processed concurrently. Even if you increase the database server's multiprogramming level, requests may need to wait for memory to become available. See "The memory governor" [*SQL Anywhere Server - SQL Usage*].

- **Memory for data structures** The database server uses resources to parse and optimize statements. For very complex statements or small cache sizes, the memory consumed for server data structures can exceed the amount that is available. A memory governor limits the amount of memory used for each task's server data structures. Each task has the following approximate limit:

  ```
  (3/4 maximum-cache-size) / (number-of-active-requests)
  ```

  If this limit is exceeded, the statement fails with an error.

Lowering the multiprogramming level

Reducing the database server's multiprogramming level by lowering the number of concurrently-executing tasks usually lowers the database server's throughput. However, lowering the

multiprogramming level may improve the response time of individual requests because there are fewer requests to compete for resources, and there is a lower probability of lock contention.

When the multiprogramming level is set too low, thread deadlock can occur. If the multiprogramming level is at a reasonable level for the workload, the occurrence of thread deadlock is symptomatic of an application design problem that results in substantial contention, and as a result, impairs scalability. One example is a table that every application must modify when inserting new data to the database. This technique is often used as part of a scheme to generate primary keys. However, the result is that it effectively serializes all the application's insert transactions. When the rate of insert transactions becomes higher than what the server can service because of the serialization on the shared table, thread deadlock usually occurs. See "Deadlock" [*SQL Anywhere Server - SQL Usage*].

The -gnl database server option sets the minimum multiprogramming level. See "-gnl dbsrv12 server option" on page 192.

See also

- "-gn dbsrv12 server option" on page 189
- "-gna dbsrv12 server option" on page 190
- "-gnh dbsrv12 server option" on page 190
- "-gnl dbsrv12 server option" on page 192
- "-gns dbsrv12 server option" on page 192
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Performance Statistics utility (dbstats) (Unix)" on page 824
- "Monitor statistics using Windows Performance Monitor" [*SQL Anywhere Server - SQL Usage*]
- "Connection, database, and database server properties" on page 619

Adjust the multiprogramming level manually

You can adjust a network server's multiprogramming level manually. However, if auto tuning of the multiprogramming level is enabled, then the server may re-adjust the multiprogramming level automatically sometime after the manual setting. If you lower the multiprogramming level, the new setting may not take effect immediately, as the server may have to wait for active tasks to complete before the new multiprogramming level can take effect. If you attempt to set the multiprogramming level to a value outside the lower and upper bounds as specified by the MinMultiProgrammingLevel and MaxMultiProgrammingLevel settings, then an error will be returned.

You can disable automatic multiprogramming level tuning either when you start the database server or when the database server is running:

- **Disabling automatic multiprogramming tuning at server startup** When you start the database server, specify -gna 0 to disable automatic tuning of the multiprogramming level:

  ```
  dbsrv12 -gna 0 ...
  ```

  You can change the default multiprogramming level using the -gn option, and set the minimum and maximum values using the -gnl and -gnh options, respectively.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Disabling automatic multiprogramming tuning of a running server** If the database server is already running, execute the following SQL statement to disable automatic tuning of the multiprogramming level:

  ```
  CALL sa_server_option ( 'AutoMultiProgrammingLevel', 'NO' );
  ```

  You can set the minimum and maximum values using the MinMultiProgrammingLevel and MaxMultiProgrammingLevel properties, respectively. For example:

  ```
  CALL sa_server_option ( 'MinMultiProgrammingLevel', 10 );
  CALL sa_server_option ( 'MaxMultiProgrammingLevel', 100 );
  ```

  You can set the current multiprogramming level using the CurrentMultiProgrammingLevel property as follows:

  ```
  CALL sa_server_option ( 'CurrentMultiProgrammingLevel', 25 );
  ```

See also

- "-gna dbsrv12 server option" on page 190
- "-gn dbsrv12 server option" on page 189
- "-gnh dbsrv12 server option" on page 190
- "-gnl dbsrv12 server option" on page 192
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

Running the database server outside the current session

When you log on to a computer using a user ID and a password, you establish a **session**. When you start a database server, or any other application, it runs within that session. When you log off the computer, all applications associated with the session shut down.

If you need the database server to be available all the time, you can run SQL Anywhere for Windows and for Unix so that when you log off the computer, the database server remains running.

- **Windows service** You can run the Windows database server as a service. This configuration can be convenient for running high availability servers. See "Understanding Windows services" on page 58.
- **Unix daemon** You can run the Unix database server as a daemon using the -ud option, enabling the database server to run in the background, and to continue running after you log off. See "Running the Unix database server as a daemon" on page 67.
- **Linux service** You can run the Linux database server as a service. This configuration has many convenient properties for running high availability servers. See "Service utility (dbsvc) for Linux" on page 836.

In addition to creating services for SQL Anywhere database servers, you can also create Windows services for the following executables:

- SQL Remote Message Agent (dbremote)
- MobiLink server (mlsrv12)
- MobiLink synchronization client (dbmlsync)
- rshost utility (rshost)
- RSOE
- SQL Anywhere Broadcast Repeater (dbns12)
- Listener utility (dblsn)

See also

- "Service utility (dbsvc) for Windows" on page 840

Understanding Windows services

You can run the database server like a Microsoft Windows program rather than a service. However, there are limitations running it as a standard program and in multi-user environments.

Limitations of running as a standard executable

When you start a program, it runs under your Windows login session, which means that if you log off the computer, the program shuts down. This configuration restricts the use of the computer if you want to keep a program running most of the time, as is commonly the case with database servers. You must stay logged on to the computer running the database server for the database server to keep running. This configuration can also present a security risk as the Windows computer must be left in a logged on state.

Advantages of services

Installing an application as a Windows service enables it to run even when you log off.

When you start a service, it logs on using a special system account called LocalSystem (or another account that you specify). Since the service is not tied to the user ID of the person starting it, the service remains open even when the person who started it logs off. You can also configure a service to start automatically when the Windows computer starts, before a user logs on.

Managing services

Sybase Central provides a more convenient and comprehensive way of managing SQL Anywhere services than the Windows services manager. You can also use the dbsvc utility to create and modify services. See "Service utility (dbsvc) for Windows" on page 840.

See also

- "Managing Windows services" on page 59

Programs that can be run as Windows services

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can run the following programs as services:

- Network database server (*dbsrv12.exe*)
- Personal database server (*dbeng12.exe*)
- SQL Remote Message Agent (*dbremote.exe*)
- MobiLink server (*mlsrv12.exe*)
- MobiLink synchronization client (*dbmlsync.exe*)
- MobiLink Relay Server (*rshost.exe*)
- MobiLink Relay Server Outbound Enabler (*rsoe.exe*)
- MobiLink Listener utility (*dblsn.exe*)
- SQL Anywhere Broadcast Repeater utility (*dbns12.exe*)
- SQL Anywhere Volume Shadow Copy Service (*dbvss12.exe*)

Not all of these applications are supplied in all editions of SQL Anywhere.

See also

- "Create Windows services" on page 59

Managing Windows services

You can perform the following Windows service management tasks from the command line, or on the Services tab in Sybase Central:

- Create, edit, and delete services
- Start and stop services
- Modify the parameters governing a service
- Add databases to a service so that you can run several databases at one time

The service icons in Sybase Central display the current state of each service using an icon that indicates whether the service is running or stopped.

Create Windows services

This section describes how to set up services using Sybase Central and the Service utility.

To create a new service (Sybase Central)

1. In the left pane, select **SQL Anywhere 12**.
2. In the right pane, click the **Services** tab.
3. Choose **File** » **New** » **Service**.
4. Follow the instructions in the **Create Service Wizard**.

Tip
You can also create services for the MobiLink plug-in. See "Running the MobiLink server outside the current session" [*MobiLink - Server Administration*].

To create a new service (command line)

- Run a dbsvc command that includes the -w option.

  For example, to create a personal server service called myserv where the database server runs as the LocalSystem user, enter the following command:

  ```
  dbsvc -as -w myserv "C:\Program Files\SQL Anywhere 12\Bin32\dbeng12.exe"
   -n william -c 8m "c:\temp\sample.db"
  ```

See "Service utility (dbsvc) for Windows" on page 840.

Notes

- Service names must be unique within the first eight characters.
- If you choose to start a service automatically, it starts whenever the computer starts Windows. If you choose to start the service manually, you need to start the service from Sybase Central each time. You may want to select **Disabled** if you are setting up a service for future use.
- When creating a service in Sybase Central, type options for the executable, without the executable name itself, in the window. For example, if you want a network server to run using the sample database with a cache size of 20 MB and the name myserver, you would type the following in the **Parameters** text box of the **Create Service Wizard** in Sybase Central:

  ```
  -c 20M
  -n myserver samples-dir\demo.db
  ```

  Line breaks are optional.
- Choose the account under which the service will run: the special LocalSystem account or another user ID.

  For more information about this choice, see "Setting the account options" on page 63.
- If you want the service to be accessible from the Windows desktop, select **Allow Service To Interact With Desktop**. If this option is cleared, no icon appears in the system tray and neither do any windows appear on the desktop.

See "Configure Windows services" on page 61.

Delete Windows services

Deleting a service removes the service name from the list of services. Deleting a service does not remove any software from your hard disk.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

If you want to re-install a service you previously deleting, you need to re-type the options. You cannot delete a service while it is running.

To delete a Windows service (Sybase Central)

1. In the left pane, select **SQL Anywhere 12**.

   In the right pane, click the **Services** tab.

2. In the right pane, select the service you want to remove and from the **File** menu, choose **Delete**.

To delete a Windows service (command line)

- Run the dbsvc utility with the -d option.

  For example, to delete the service called myserv, without prompting for confirmation, type the following command:

  ```
  dbsvc -y -d myserv
  ```

See also

- "Service utility (dbsvc) for Windows" on page 840

Configure Windows services

A service runs a database server or other application with a set of options.

In addition to the options, services accept other parameters that specify the account under which the service runs and the conditions under which it starts.

To change the parameters for a service (Sybase Central)

1. In the left pane, select **SQL Anywhere 12**.
2. In the right pane, select the service you want to change.
3. From the **File** menu, choose **Properties**.
4. Alter the parameters as needed on the tabs of the **Service Properties** window.
5. Click **OK** when finished.

Changes to a service configuration take effect the next time someone starts the service. The **Startup** option is applied the next time Windows is started.

Setting Windows service startup options

The following options govern startup behavior for SQL Anywhere services. You can set them on the **General** tab of the **Service Properties** window.

- **Automatic** If you choose the **Automatic** setting, the service starts whenever the Windows operating system starts. This setting is appropriate for database servers and other applications running all the time.
- **Manual** If you choose the **Manual** setting, the service starts only when a user with Administrator permissions starts it. For information about Administrator permissions, see your Windows documentation.
- **Disabled** If you choose the **Disabled** setting, the service does not start.

Specifying options for Windows services

The options for a service are the same as those for the executable.

Caution
The **Configuration** tab of the **Service Properties** window provides a **Parameters** text box for specifying options for a service. Do not type the name of the program executable in this box.

Examples

To start a network server service with a cache size of 20 MB, named my_server running two databases, you would type the following in the **Parameters** field:

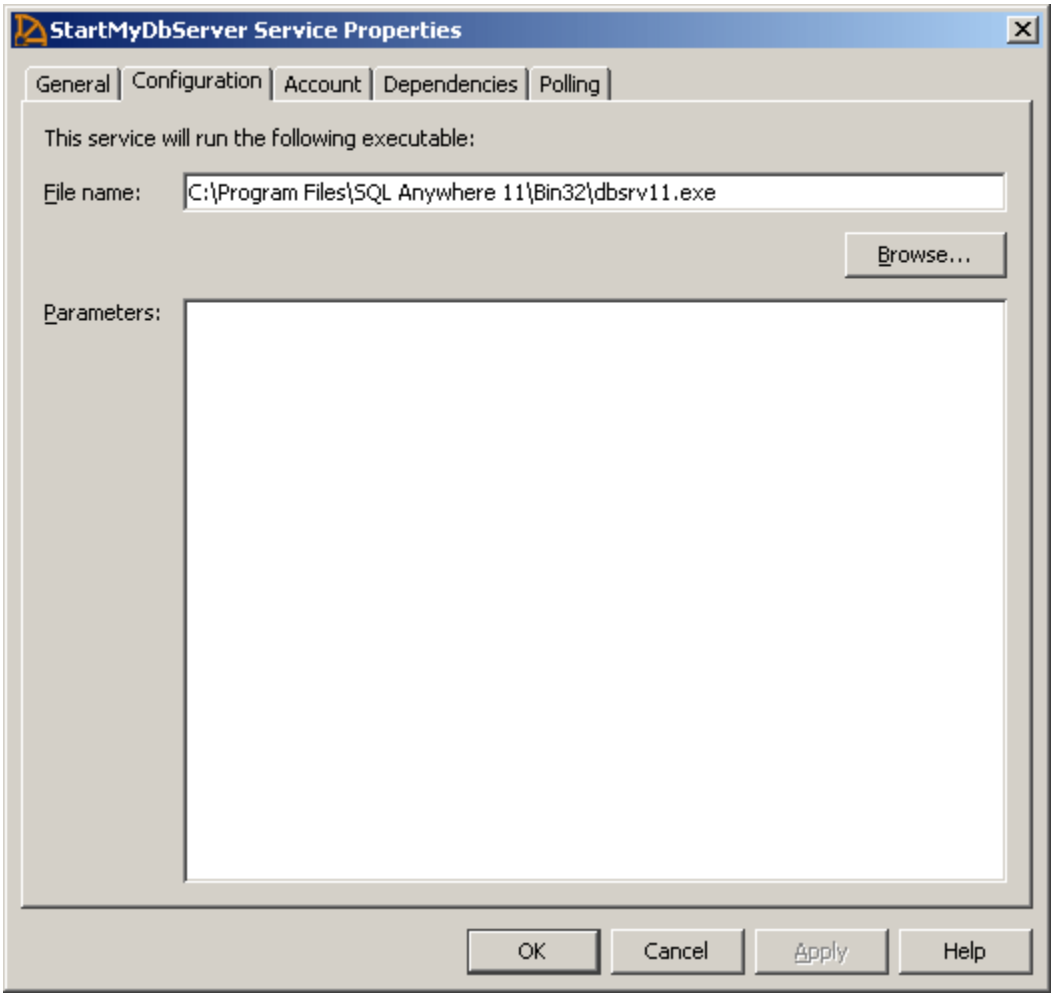
```
-c 20M
-n my_server
c:\db_1.db
c:\db_2.db
```

To start a SQL Remote Message Agent service connecting to the sample database as user ID DBA, you would type the following:

```
-c "UID=DBA;PWD=sql;DBN=demo"
```

The following figure illustrates a sample **Service Properties** window.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0



Setting the account options

You can choose under which account the service runs. Most services run under the special LocalSystem account, which is the default option for services. You can set the service to log on under another account by opening the **Account** tab on the **Service Properties** window, and typing the account information.

If you choose to run the service under an account other than LocalSystem, that account must have the **Log On As A Service** privilege. This privilege can be granted from the Windows User Manager application under Advanced Privileges. The Service utility (dbsvc) also grants this privilege if it is required.

When an icon appears in the system tray

- If a service runs under LocalSystem, and **Allow Service To Interact With Desktop** is selected on the **Service Properties** window, an icon appears on the desktop of every user logged in to Windows on

the computer running the service. Any user can open the application window and stop the program running as a service.

- If a service runs under LocalSystem, and **Allow Service To Interact With Desktop** is cleared on the **Service Properties** window, no icon appears on the desktop for any user. Only users with permissions to change the state of services can stop the service.
- If a service runs under another account, no icon appears on the desktop. Only users with permissions to change the state of services can stop the service.

Changing the executable file

To change the program executable file associated with a service, click the **Configuration** tab on the **Service Properties** window and type the new path and file name in the File Name text box.

If you move an executable file to a new directory, you must modify this entry.

Add new databases to a service

Each network server or personal server can run more than one database. If you want to run more than one database at a time, it is recommended that you do so by attaching new databases to your existing service, rather than by creating new services.

To add a new database to an existing service (Sybase Central)

1. From the **Context** dropdown list, choose **SQL Anywhere 12**.
2. In the right pane, click the **Services** tab.
3. Select the service and then from the **File** menu, choose **Properties**.
4. Click the **Configuration** tab.
5. Add the path and file name of the new database to the end of the list of options in the **Parameters** box.
6. Click **OK** to save the changes.

   The new database starts the next time the service starts.

Databases can be started on running servers by client applications, such as Interactive SQL.

For more information about starting a database from Interactive SQL, see “START DATABASE statement” [*SQL Anywhere Server - SQL Reference*].

For more information about how to implement this function in an embedded SQL application, see “db_start_database function” [*SQL Anywhere Server - Programming*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Starting a database from an application does not attach it to the service. If the service is stopped and restarted, the additional database will not be started automatically.

Set the refresh frequency

By default, Sybase Central checks the status (started or stopped) of each Windows service, and updates the icons to display the current state. In addition Sybase Central refreshes the dynamic objects such as connections and locks, as well as dynamic properties of events and maintenance plans. By default, Sybase Central refreshes every 10 seconds. If you turn off automatic refreshing, you must click **View** » **Refresh Folder** to see changes to the state.

To set the refresh frequency (Sybase Central)

1. Choose **Tools** » **SQL Anywhere 12** » **Preferences**.
2. Click the **Automatic Refresh** tab.
3. Select **Enable Automatic Refreshing**.
4. Specify a time interval in the **Refresh Every *X* seconds**. The default is 10 seconds.

   The value you set in this window remains in effect for subsequent sessions until you change it.
5. Click **Apply**.
6. Click **OK**.

Start and stop services

To start or stop a service (Sybase Central)

1. From the **Context** dropdown list, choose **SQL Anywhere 12**.
2. In the right pane, click the **Services** tab.
3. Select the service and then from the **File** menu, choose **Start** or **Stop**.

If you start a service, it keeps running until you stop it. Closing Sybase Central or logging off does not stop the service.

Stopping a database server service closes all connections to the database and stops the database server. For other applications, the program shuts down.

The Windows Service Manager

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can use Sybase Central to perform all the service management for SQL Anywhere. Although you can use the Windows **Service Manager** in the **Control Panel** for some tasks, you cannot install or configure a SQL Anywhere service from the Windows **Service Manager**.

If you open the Windows **Service Manager** (from the Windows **Control Panel**), a list of services appears. The names of the SQL Anywhere services are formed from the service name you provided when installing the service, prefixed by SQL Anywhere. All the installed services appear together in the list.

Service dependencies

In some circumstances you may want to run more than one executable as a service, and these executables may depend on each other. For example, you may want to run a server and a SQL Remote Message Agent to assist in replication.

Services must start in the correct order. If a SQL Remote Message Agent service starts before the server has started, it fails because it cannot find the server.

You can prevent these problems by using **service groups**, which you manage from Sybase Central.

Service groups overview

You can assign each service on your system to be a member of a service group. By default, each service belongs to a group, as listed in the following table.

Service	Default group
Network server	SQLANYServer
Personal server	SQLANYEngine
MobiLink synchronization client	SQLANYMLSync
SQL Remote Message Agent	SQLANYRemote
MobiLink server	SQLANYMobiLink
Broadcast Repeater utility	SQLANYNS
MobiLink Listener	SQLANYLSN
SQL Anywhere Volume Shadow Copy Service	SQLANYVSS

Before you can configure your services to ensure that they start in the correct order, you must check that your service is a member of an appropriate group. You can check which group a service belongs to, and change this group, from Sybase Central.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To check and change which group a service belongs to (Sybase Central)

1. From the **Context** dropdown list, choose **SQL Anywhere 12**.
2. In the right pane, click the **Services** tab.
3. Select the service and then from the **File** menu, choose **Properties**.
4. Click the **Dependencies** tab. The top text box displays the name of the group the service belongs to.
5. Click **Change** to display a list of available groups on your system.
6. Select one of the groups, or type a name for a new group.
7. Click **OK** to assign the service to that group.

Manage service dependencies

With Sybase Central you can specify **dependencies** for a service. For example:

- You can ensure that at least one member of each of a list of service groups has started before the current service.
- You can ensure that any number of services start before the current service. For example, you may want to ensure that a particular network server has started before a SQL Remote Message Agent that is to run against that server starts.

To add a service or group to a list of dependencies (Sybase Central)

1. From the **Context** list, choose **SQL Anywhere 12**.
2. In the right pane, click the **Services** tab.
3. Select the service, and then choose **File** » **Properties**.
4. Click the **Dependencies** tab.
5. Click **Add Services** or **Add Service Groups** to add a service or group to the list of dependencies.
6. Select one of the services or groups from the list.
7. Click **OK** to add the service or group to the list of dependencies.

Running the Unix database server as a daemon

To run the Unix database server in the background, and to enable it to run independently of the current session, you run it as a **daemon**.

Do not use '&' to run the database server in the background
If you use the Unix & (ampersand) command to run the database server in the background, it will not work—the server will either immediately shut down or stop responding. You must instead run the database server as a daemon.

As well, attempting to start a server in the background from within a program using the typical `fork()-exec()` sequence will not work. If you need to do this, add the -ud option to the list of database server options.

You can run the Unix database server as a daemon in one of the following ways:

1. Use the -ud option when starting the database server. For example:

   ```
   dbsrv12 -ud demo
   ```

2. Use the dbspawn tool to start the database server. For example:

   ```
   dbspawn dbsrv12 demo
   ```

   One advantage of using dbspawn is that the dbspawn process does not shut down until it has confirmed that the daemon has started and is ready to accept requests. If for any reason the daemon fails to start, the exit code for dbspawn will be non-zero.

   When you start the daemon directly using the -ud option, the dbeng12 and dbsrv12 commands create the daemon process and return immediately (exiting and allowing the next command to be executed) before the daemon initializes itself or attempts to open any of the databases specified in the command.

   If you want to ensure that a daemon is running one or more applications that will use the database server, you can use dbspawn to ensure that the daemon is running before starting the applications. The following is an example of how to test this using a csh script.

   ```
   #!/bin/csh
   # start the server as a daemon and ensure that it is
   # running before you start any applications
   dbspawn dbsrv12 demo
   if ( $status != 0 ) then
      echo Failed to start demo server
      exit
   endif
   # ok, now you can start the applications
   ...
   ```

   This example uses an sh script to test whether the daemon is running before starting the applications.

   ```
   #!/bin/sh
   # start the server as a daemon and ensure that it is
   # running before you start any applications
   dbspawn dbsrv12 demo
   if [ $? != 0 ]; then
      echo Failed to start demo server
      exit
   fi
   # ok, now you can start the applications
   ...
   ```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

3. Spawn a daemon from within a C program, for example:

```
...
if( fork() == 0 ) {
      /* child process = start server daemon */
      execl( "/opt/sqlanywhere12/bin/dbsrv12",
"dbsrv12", "-ud", "demo" );
   exit(1);
}
/* parent process */
...
```

   Note that the -ud option is used.

See also

- "-ud dbeng12/dbsrv12 server option" on page 230
- "Start Server in Background utility (dbspawn)" on page 849

Running authenticated SQL Anywhere applications

The OEM Edition of SQL Anywhere is provided for Sybase OEM Partners. With the OEM Edition of SQL Anywhere, an authenticated application can carry out any operation on the database, subject to the permissions granted to the user ID.

Unauthenticated connections have read-only access, and can perform inserts, updates, and deletes on temporary tables. Using unauthenticated connections allows complex reports to be created using stored procedures, and accessed using reporting tools, such as Crystal Reports.

The authentication mechanism is independent of any application programming language or tool, and is carried out on every connection, so you can use both authenticated connections and more restricted unauthenticated connections in your application.

Authentication is not a security mechanism. Anyone running an unauthenticated database server against the database can carry out any operation, subject to the usual SQL permissions scheme.

Developing an authenticated application

Developing an authenticated application is a simple process: a special authentication signature is incorporated into the database, and a second signature is incorporated into your application. When the application connects to the database, the signatures are compared to authenticate the application. The following steps are required to develop an authenticated SQL Anywhere application:

1. "Obtaining authentication signatures" on page 70
2. "Authenticating your database" on page 70
3. "Authenticating your application" on page 71

All the database tools included with SQL Anywhere, including Sybase Central, Interactive SQL, and the utilities, such as dbbackup, are self-authenticating. They are unrestricted in their operations against any authenticated database. If the database itself is not authenticated, the tools act in a restricted, read-only fashion.

You must use the OEM Edition of the SQL Anywhere database server for an authenticated application. This edition differs from the usual database server only in that it processes authentication instructions. The authentication instructions are ignored by other editions of the database server. If you do not use the authenticated database server, no restrictions are placed on unauthenticated applications.

Obtaining authentication signatures

Note
To get an authentication signature, you must have an OEM contract with Sybase iAnywhere.

To obtain your authentication signature

1. Go to http://www.sybase.com/sql_anywhere_authentication_registration.
2. Complete the form to obtain your authentication signatures. The following information is incorporated into your authentication mechanism:
   - **Company** The name of your company.
   - **Application Name** The name of your application.

For information about how the company name and application name are incorporated into the authentication mechanism, see “Authenticating your database” on page 70.

Once you complete the form, you will be emailed a database signature and an application signature within 48 hours. These signatures are long strings of characters and digits. The email message containing your authentication information includes some examples of how to use the information. Some email systems force line breaks in these instructions. Make sure you rejoin lines broken in the email message for the instructions to work.

Authenticating your database

The OEM Edition of SQL Anywhere does not permit any operations to be carried out on an unauthenticated database.

You can use the Authenticated database property to determine if the database has been authenticated:

```
SELECT DB_PROPERTY ( 'Authenticated' );
```

For more information about database properties, see “Database properties” on page 659.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To authenticate a database

1. Set the database_authentication option for the database, using the following SQL authentication statement:

```
SET OPTION PUBLIC.database_authentication='company=company-name;
     application=application-name;
     signature=database-signature';
```

> **Note**
> Line breaks have been added to the syntax example to improve readability. However, the syntax should be executed without line breaks and without spaces between the equal sign and semi-colons.

2. The *company-name* and *application-name* arguments are the values you supplied to Sybase when obtaining your signature, and *database-signature* is the database signature that you received from Sybase.

3. Restart the database for the option to take effect.

When the database server loads an authenticated database, it displays a message in the database server messages window describing the authenticated company and application. You can check that this message is present to verify that the database_authentication option has taken effect. The message has the following form:

```
This database is licensed for use with:
Application: application-name
Company: company-name
```

> **Tip**
> You can store the authentication statement in a SQL script file to avoid having to type in the long signature. You can run the SQL script from Interactive SQL by choosing **Run Script** from the **File** menu.
>
> If you create a file named *authenticate.sql* in the *scripts* subdirectory of your SQL Anywhere installation directory and store the authentication statement in this file, it is applied whenever you create, rebuild, or upgrade a database. See "Upgrading authenticated databases" on page 74.

Authenticating your application

An authenticated application must set the connection_authentication database option immediately after connecting. The option must be set on every connection immediately after the connection is established. ODBC or JDBC applications query the database about its capabilities, and you may not have control over these actions. For this reason, every connection has a thirty second grace period before the restrictions apply. The grace period allows an application to authenticate regardless of which development tool is being used.

You can use the Authenticated connection property to determine if the database has been authenticated:

```
SELECT CONNECTION_PROPERTY ( 'Authenticated' );
```

For more information about connection properties, see "Connection properties" on page 619.

The following SQL statement authenticates the connection:

```
SET TEMPORARY OPTION connection_authentication='company = company-name;
     application=application-name;
     signature=application-signature';
```

Note
Line breaks have been added to the syntax example to improve readability. However, the syntax should be executed without line breaks and without spaces between the equal sign and semi-colons.

The option must be set for the duration of the connection only by using the TEMPORARY keyword. The *company-name* and *application-name* must match those in the database authentication statement. The *application-signature* is the signature that you obtained from Sybase.

The database server verifies the application signature against the database signature. If the signature is verified, the connection is authenticated and has no restrictions on its activities beyond those imposed by the SQL permissions. If the signature is not verified, the connection is limited to those actions permitted by unauthenticated applications.

Executing the authentication statement

The way you execute the SET TEMPORARY OPTION statement that sets the authentication option depends on the programming interface you are using. The signatures listed here are not valid signatures. Examples are provided for setting the authentication option using the following interfaces:

- ODBC
- Sybase PowerBuilder
- JDBC
- ADO.NET
- Embedded SQL

Using special characters in the authentication option
If your company name has quotation marks, apostrophes, or other special characters (for example, Joe's Garage) you need to be careful about how you construct the authentication statement. The entire set of authentication options (Company=...;Application=...;Signature=...) is a SQL string. The rules for strings in SQL dictate that if you include a quotation mark inside the string, it must be doubled to be accepted. For example:

```
SET TEMPORARY OPTION connection_authentication='Company = Joe''s Garage;
    Application=Joe''s Program;
    Signature=0fa55157edb8e14d818e...';
```

Line breaks have been added to the syntax example to improve readability. However, the syntax should be executed without line breaks and without spaces between the equal sign and semi-colons.

ODBC

Use the following statement:

```
SQLExecDirect(
    hstmt,
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
    "SET TEMPORARY OPTION connection_authentication=
       'Company=MyCo;
        Application=MyApp;
        Signature=0fa55159999e14d818e...';",
    SQL_NTS
);
```

The string must be entered on a single line, or you must build it up by concatenation.

Sybase PowerBuilder

Use the following PowerScript statement:

```
EXECUTE IMMEDIATE
   "SET TEMPORARY OPTION connection_authentication=
       'Company=MyCo;
        Application=MyApp;
        Signature=0fa551599998e14d818e...';"
USING SQLCA
```

JDBC

Use the following statement:

```
Statement Stmt1 = con.createStatement();
Stmt1.executeUpdate(
    "SET TEMPORARY OPTION connection_authentication=
    'Company=MyCo;
     Application=MyApp;
     Signature=0fa55159999e14d818e...';"
);
```

The string must be entered on a single line, or you must build it up by concatenation.

ADO.NET

Use the following statement:

```
SACommand cmd=new SACommand(
   "SET TEMPORARY OPTION connection_authentication=
       'Company=MyCo;
        Application=MyApp;
        Signature=0fa551599998e14d818e...';",
    con
);
cmd.ExecuteNonQuery();
```

The string must be entered on a single line, or you must build it up by concatenation.

Embedded SQL

```
Use the following statement:
EXEC SQL SET TEMPORARY OPTION connection_authentication=
   'Company=MyCo;
    Application=MyApp;
    Signature=0fa551599998e14d818e...';
```

The string must be entered on a single line, or you must build it up by concatenation.

When connecting to an authenticated database, the connection and authentication steps are performed separately. However, some objects, such as the Microsoft Visual Basic Grid object can attempt a separate,

implicit connection, which does not automatically include authentication. In such cases, the connection is not authenticated and the database operation can fail. You can avoid this problem by including the InitString connection parameter in the connection string. The following example illustrates how you can modify a Microsoft Visual Basic application to include the InitString connection parameter so that every connection is immediately followed by authentication:

```
mConnectionString =
   "Provider=SAPROV.12;
    UID=DBA;
    PWD=sql;
    Host=test12;
    InitString=SET TEMPORARY OPTION connection_authentication=
       'Company=MyCo;
        Application=MyApp;
        Signature=0fa55157edb8e14d818e...'"
mdbName.ConnectionString = mConnectionString
mdbName.Open
mIsSQL = True
```

Upgrading authenticated databases

The only way to preserve authentication information when upgrading or rebuilding a database is to store the authentication statement in the file *authenticate.sql*.

The CREATE DATABASE and ALTER DATABASE UPGRADE statements, dbupgrad and dbunload utilities, and the **Upgrade Database Wizard** check for the existence of a file named *authenticate.sql* in the *install-dir/scripts* directory and run its contents if it exists. You must create this file before running the upgrade or rebuild on the database.

To upgrade or rebuild an authenticated database

1. Create a file named *authenticate.sql* in the *install-dir\scripts* directory.

2. Add the following contents to the file:

   ```
   SET OPTION PUBLIC.database_authentication = 'authentication-statement'
   go
   ```

   The go must appear in the file; otherwise, the statement is ignored.

   For information about the content of the *authentication-statement* string, see "database_authentication" on page 527.

3. Upgrade or rebuild the database. The steps you must follow depend on the version of the database file you are upgrading or rebuilding. See:

   - "Rebuilding version 9 and earlier databases for version 12" [*SQL Anywhere 12 - Changes and Upgrading*]
   - "Upgrading version 10 and later databases" [*SQL Anywhere 12 - Changes and Upgrading*]
   - "Rebuilding version 10 and later databases" [*SQL Anywhere 12 - Changes and Upgrading*]

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Running SQL Anywhere on Windows Vista, Windows 7, and Windows 2008

The following considerations apply when running SQL Anywhere software on Vista, Windows 7, and Windows 2008:

- **Security** These operating systems incorporate a security model called User Account Control (UAC). UAC is enabled by default and may affect the behavior of programs that expect to be able to write files, especially when the computer supports more than one user. Depending on where and how files and directories are created, a file created by one user may have permissions that do not allow another user to read or write to that file. If you install SQL Anywhere into the default directories, then files and directories that require read/write access for multiple users are set up appropriately.

- **SQL Anywhere elevated operations agent** Certain actions require privilege elevation to execute when run under UAC. The following programs may require elevation in SQL Anywhere:

  - *dbdsn.exe*
  - *dbelevate12.exe*
  - *dblic.exe*
  - *dbsvc.exe*
  - *installULNet.exe*
  - *mlasinst.exe*
  - *SetupVSPackage.exe*
  - *ulcond12.exe*

  The following DLLs require elevation when they are registered or unregistered:

  - *dbctrs12.dll*
  - *dbodbc12.dll*
  - *dboledb12.dll*
  - *dboledba12.dll*

  If UAC is activated, you may receive an elevation prompt for the SQL Anywhere elevated operations agent. The prompt is issued by the User Account Control system to confirm that you want to continue running the identified program (if logged on as an administrator) or to provide administrator credentials (if logged on as a non-administrator).

- **Deployment considerations** The program *dbelevate12.exe* is used internally by SQL Anywhere components to perform operations that require elevated privileges. This executable must be included in deployments of SQL Anywhere.

- **Microsoft ActiveSync support** The Microsoft ActiveSync utility is not supported in Vista, Windows 7 or Windows 2008. It is replaced by the Windows Mobile Device Center. You can use the SQL Anywhere ActiveSync Provider Installation utility with Windows Mobile Device Center.

- **SQL Anywhere executables signed** SQL Anywhere executables are signed by iAnywhere Solutions, Inc.

- **Windows services** Services that are compliant with Windows Vista, Windows 7, and Windows 2008 are not allowed to interact with the desktop. On these operating systems, no SQL Anywhere services interact with the desktop (even if **Allow Service To Interact With Desktop** is enabled in the service definition). SQL Anywhere database servers can be monitored from Sybase Central or the dbconsole utility. See "SQL Anywhere Console utility (dbconsole)" on page 848.

  Sybase Central disables the option to allow services to interact with desktop when running on these operating systems.

- **Using an AWE cache** To use an AWE cache, you must run the database server as administrator. Starting a non-elevated database server with an AWE cache results in a warning that the database server must be run as an administrator to use AWE. See "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171.

  Note
  The use of AWE is deprecated. It is recommended that you use the 64-bit version of the SQL Anywhere database server on a 64-bit Windows operating system if you require a large cache.

Client/server communications

Database servers communicate over a variety of protocols. SQL Anywhere includes many options for customizing the behavior of supported protocols. This section also includes information about troubleshooting SQL Anywhere connections.

For information about HTTP and HTTPS connections, see "Using SQL Anywhere as an HTTP web server" [*SQL Anywhere Server - Programming*] and "Accessing web services using web clients" [*SQL Anywhere Server - Programming*].

For information about using transport-layer security, see "Transport-layer security" on page 1143.

Selecting communications protocols

Any communication between a client application and a database server requires a communications protocol. SQL Anywhere supports communications protocols for communications across networks and for same-computer communications.

Available protocols for the personal database server

By default, the personal database server starts only the shared memory protocol.

The personal database server (*dbeng12.exe*) supports the following protocols:

- **Shared memory** This protocol is for same-computer communications, and always remains available. It is available on all platforms.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For same-computer communications, shared memory tends to provide better performance than TCP/IP.

- **TCP/IP** This protocol is for same-computer communications from TDS clients, Sybase Open Client, or the jConnect JDBC driver. You must use TCP/IP if you want to connect from Sybase Open Client or jConnect. It is available on all platforms.

  For more information about TDS clients, see "Using SQL Anywhere as an Open Server" on page 1165.

Available protocols for the network server

By default, the network database server starts all available protocols. You can limit the protocols available to a database server using the -x option. See "-x dbeng12/dbsrv12 server option" on page 236.

The network database server (*dbsrv12.exe*) supports the following protocols:

- **Shared memory** This protocol is for same-computer communications, and always remains available. It is available on all platforms.
- **TCP/IP** This protocol is used for communications between different computers, and you must use TCP/IP if you are using TDS clients, Sybase Open Client, or the jConnect JDBC driver. It is available on all platforms.

Shared memory and terminal services
When using terminal services, shared memory clients can only find database servers running in the same terminal. If you use terminal services with a database server that is running as a service, only clients running on the console can connect. Clients running on non-console terminals cannot connect over shared memory. In these situations, you can use TCP/IP instead of shared memory to allow clients to connect.

For information about securing shared memory connections on Unix, see "Security tips" on page 1116.

See also

- "Using the TCP/IP protocol" on page 78

Specifying protocols

By specifying the -x option, you can instruct a database server to use only some of the available network protocols. The following command starts the sample database on the network database server using the TCP/IP protocol:

```
dbsrv12 -x "tcpip" samples-dir\demo.db
```

The following command starts the sample database on the personal database server using the TCP/IP protocol:

```
dbeng12 -x "tcpip" samples-dir\demo.db
```

Although not strictly required in this example, the quotation marks are necessary if there are spaces in any of the arguments to -x.

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can add additional parameters to tune the behavior of the server for each protocol. For example, the following command (typed all on one line) instructs the server to use two network cards.

```
dbsrv12 -x "tcpip(MyIP=192.75.209.12,192.75.209.32)" samples-dir\demo.db
```

For more information about available network protocol options that can be used with the -x option, see "Network protocol options" on page 311.

Using the TCP/IP protocol

TCP/IP is a suite of protocols that is used for connecting to databases that are running on different computers.

UDP is a transport layer protocol that sits on top of IP. SQL Anywhere uses UDP on top of IP to do initial server name resolution and uses TCP for connection and communication after that.

When you use the TCP/IP protocol, you can secure client/server communications using transport-layer security and ECC or RSA encryption technology. See "Transport-layer security" on page 1143.

The client library for each platform supports the same protocols as the corresponding database server. For SQL Anywhere to run properly, the network protocol (TCP/IP) must be installed and configured properly on both the client and server computers.

Using TCP/IP with Windows

The TCP/IP implementation for database servers on all Windows platforms uses Winsock 2.2. Clients on Windows Mobile use the Winsock 1.1 standard.

If you do not have TCP/IP installed, you can install the TCP/IP protocol from the **Control Panel** by double-clicking **Network Settings**.

See also

- "-x dbeng12/dbsrv12 server option" on page 236
- "Host connection parameter" on page 291
- "Broadcast (BCAST) protocol option" on page 313
- "Host (IP) protocol option" on page 321
- "MyIP (ME) protocol option" on page 333
- "Testing the TCP/IP protocol" on page 1003

Tuning TCP/IP performance

Increasing the packet size may improve query response time, especially for queries transferring a large amount of data between a client and a server process. You can set the packet size using the -p option in the database server command, or by setting the CommBufferSize (CBSIZE) connection parameter in your connection profile.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "-p dbeng12/dbsrv12 server option" on page 212
- "CommBufferSize (CBSIZE) connection parameter" on page 271

IPv6 support in SQL Anywhere

On IPv6-enabled computers, the network database server listens by default on all IPv4 and IPv6 addresses. IPv6 is supported on Windows, Linux, Mac OS X, Solaris, IBM AIX, and HP-UX.

Usually no changes are required to the database server start line to use IPv6. In the cases where specifying an IP address is required, the server and the client libraries both accept IPv4 and IPv6 addresses. For example, if a computer has more than one network card enabled, it probably has two IPv4 addresses and two IPv6 addresses. If you want the database server to listen on only one of the IPv6 addresses, you can specify an address in the following format:

```
dbsrv12 -x tcpip(MyIP=fd77:55f:5a64:52a:202:5445:5245:444f) ...
```

Similarly, if a client application needs to specify the IP address of a server, the connection string or DSN can contain the address, in the following format:

```
...;LINKS=tcpip(HOST=fe80::5445:5245:444f);...
```

Each interface is given an interface identifier, which appears at the end of an IPv6 address. For example, if *ipconfig.exe* lists the address `fe80::5445:5245:444f%7`, the interface identifier is 7. When specifying an IPv6 address on a Windows platform, the interface identifier should be used. On Unix, you can specify either an interface identifier or interface name (the interface name is the name of the interface reported by ifconfig). For example, the interface name is eth1 in the following IPv6 address: `fe80::5445:5245:444f%eth1`. An interface identifier is required when specifying IPv6 addresses on Linux (kernel 2.6.13 and later). This requirement affects values specified by the following protocol options:

- Broadcast
- Host
- MyIP

For example, suppose *ipconfig.exe* lists two interfaces, one with the identifier 1 and the other 2. If you are looking for a database server that is on the network used by interface number 2, you can tell the client library to broadcast only on that interface:

```
LINKS=tcpip(BROADCAST=ff02::1%2)
```

Note that `ff02::1` is the IPv6 link-local multicast address.

Encrypting client/server communications over TCP/IP

By default, communication packets are not encrypted, which poses a potential security risk. You can secure communications between client applications and the database server over TCP/IP using simple

encryption or transport-layer security. Transport-layer security provides server authentication, strong encryption using ECC or RSA encryption technology, and other features for protecting data integrity. See "Transport-layer security" on page 1143.

Connecting across a firewall

There are restrictions on connections when the client application is on one side of a firewall, and the server is on the other. Firewall software filters network packets according to network port. Also, it is common to disallow UDP packets from crossing the firewall.

When connecting across a firewall, you must use a set of protocol options in the CommLinks (LINKS) connection parameter of your application's connection string.

- **Host** Set this parameter to the host name on which the database server is running. You can use the short form IP.
- **ServerPort** If your database server is not using the default port of 2638, you must specify the port it is using. You can use the short form Port.
- **ClientPort** Set this parameter to a range of allowed values for the client application to use. You can use the short form CPORT. This option may not be necessary depending on the firewall's configuration.
- **DoBroadcast=NONE** Set this parameter to prevent UDP from being used when connecting to the server.

The firewall must be configured to allow TCP/IP traffic between the SQL Anywhere database server's address and all the SQL Anywhere clients' addresses. The SQL Anywhere server's address is the IP address of the computer running the SQL Anywhere server (the HOST parameter) and the SQL Anywhere server's IP port number (the ServerPort protocol option, default 2638). Each SQL Anywhere client's address consists of the IP address of the client computer, and the range of the client IP ports (the ClientPort protocol option). For the simplest configuration, all client ports can be allowed. If only specific client ports are allowed, specify a range with more ports than the maximum number of concurrent connections from each client computer, since there is a several minute timeout before a client port can be reused.

See "ClientPort (CPORT) protocol option" on page 318.

Example

The following connection string fragment restricts the client application to ports 5050 through 5060, and connects to a server named myeng running on the computer at address myhost using the server port 2020. No UDP broadcast is performed because of the DoBroadcast option.

```
Server=myeng;LINKS=tcpip(ClientPort=5050-5060;HOST=myhost;PORT=2020;DoBroadca
st=NONE)
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "CommLinks (LINKS) connection parameter" on page 272
- "ClientPort (CPORT) protocol option" on page 318
- "ServerPort (PORT) protocol option" on page 335
- "Host (IP) protocol option" on page 321
- "DoBroadcast (DOBROAD) protocol option" on page 320

Connecting on a dial-up network connection

You can use connection and protocol options to assist with connecting to a database across a dial-up link.

On the client side, you should specify the following protocol options:

- **Host parameter** You should specify the host name or IP address of the database server using the Host (IP) protocol option. See "Host (IP) protocol option" on page 321.
- **DoBroadcast parameter** If you specify the Host (IP) protocol option, there is no need to do a broadcast search for the database server. For this reason, use direct broadcasting. See "DoBroadcast (DOBROAD) protocol option" on page 320.
- **MyIP parameter** You should set **MyIP=NONE** on the client side. See "MyIP (ME) protocol option" on page 333.
- **TIMEOUT parameter** Set the Timeout (TO) protocol option to increase the time the client will wait while searching for a server. See "Timeout (TO) protocol option" on page 338.

A typical CommLinks (LINKS) connection parameter may look as follows:

```
LINKS=tcpip(MyIP=NONE;DoBroadcast=DIRECT;HOST=server_ip)
```

Connecting using an LDAP server

You can specify a central LDAP server to keep track of all database servers in an enterprise if you are operating on a Windows (except Windows Mobile) or Unix platform. When the database server registers itself with an LDAP server, clients can query the LDAP server and find the database server they are looking for, regardless of whether they are on a WAN, LAN, or going through firewalls. Clients do not need to specify an IP address (HOST=). The Server Enumeration utility (dblocate) can also use the LDAP server to find other such servers.

LDAP is only used with TCP/IP, and only on network database servers.

Using SQL Anywhere with an LDAP server on IBM AIX

To use SQL Anywhere with IBM AIX 6, you must either create links in */usr/lib* or ensure that the directory containing the LDAP libraries is included in the LIBPATH to ensure that the LDAP system libraries are found.

To create links in /usr/lib

- Run the following commands as the root user:

```
cd /usr/lib
ln -s /opt/IBM/ldap/V6.1/lib64/libibmldap.a libibmldap64.a
ln -s /opt/IBM/ldap/V6.1/lib/libibmldap.a
```

To add the directory containing the LDAP libraries to LIBPATH

1. Create links in */usr/lib* by running the following commands as the root user:

```
cd /usr/lib
ln -s /opt/IBM/ldap/V6.1/lib64/libibmldap.a libibmldap64.a
ln -s /opt/IBM/ldap/V6.1/lib/libibmldap.a
```

2. Ensure that the directory with the LDAP libraries are in the LIBPATH.

   For example, for 64-bit libraries:

```
export LIBPATH=/opt/IBM/ldap/V6.1/lib64:$LIBPATH
```

   For example, for 32-bit libraries:

```
export LIBPATH=/opt/IBM/ldap/V6.1/lib:$LIBPATH
```

Configuring the saldap.ini file

To connect using an LDAP server, a file containing information on how to find and connect to the LDAP server must be created on both the database server computer and on each client computer. By default the name of this file is *saldap.ini*, but it is configurable. If this file doesn't exist, LDAP support is silently disabled.

The file must be located in the same directory as the SQL Anywhere executables (for example, *install-dir* *bin32* on Windows) unless a full path is specified with the LDAP parameter. The file must be in the following format:

```
[LDAP]
server=computer-running-LDAP-server
port=port-number-of-LDAP-server
basedn=Base-DN
authdn=Authentication-DN
password=password-for-authdn
search_timeout=age-of-timestamps-to-be-ignored
update_timeout=frequency-of-timestamp-updates
read_authdn=read-only-authentication-domain-name
read_password=password-for-authdn
```

You can add simple encryption to obfuscate the contents of the *saldap.ini* file using the File Hiding utility (dbfhide). See "File Hiding utility (dbfhide)" on page 794.

If the name of the file is not *ldap.ini*, then you must use the LDAP parameter to specify the file name.

server The name or IP address of the computer running the LDAP server. This value is required on Unix. If this entry is missing on Windows, then Windows looks for an LDAP server running on the local domain controller.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

port The port number used by the LDAP server. The default is 389.

basedn The domain name of the subtree where the SQL Anywhere entries are stored. This value defaults to the root of the tree.

authdn The authentication domain name. The domain name must be an existing user object in the LDAP directory that has write access to the basedn. This parameter is required for the database server, and ignored on the client.

password The password for authdn. This parameter is required for the database server, and ignored on the client.

search_timeout The age of timestamps at which they are ignored by the client and/or the Server Enumeration utility (dblocate). A value of 0 disables this option so that all entries are assumed to be current. The default is 600 seconds (10 minutes).

update_timeout The frequency of timestamp updates in the LDAP directory. A value of 0 disables this option so that the database server never updates the timestamp. The default is 120 seconds (2 minutes).

read_authdn The read-only authentication domain name. The domain name must be an existing user object in the LDAP directory that has read access to the basedn. This parameter is only required if the LDAP server requires a non-anonymous binding before searching can be done. For example, this field is normally required if Active Directory is used as the LDAP server. If this parameter is missing, the bind is anonymous.

read_password The password for authdn. This parameter is only required on the client if the read_authdn parameter is specified.

How the connection is made

When the database server starts, it checks for an existing entry with the same name in the LDAP file. If one is found, it is replaced if either the location entries in LDAP match the database server attempting to start, or the timestamp field in the LDAP entry is more than 10 minutes old (the timeout value is configurable).

If neither of these entries is true, then there is another database server running with the same name as the one attempting to start, and startup fails.

To ensure that entries in LDAP are up-to-date, the database server updates a timestamp field in the LDAP entry every 2 minutes. If an entry's timestamp is older than 10 minutes, clients ignore the LDAP entry. Both of these settings are configurable.

On the client, the LDAP directory is searched before doing any broadcasting, so if the database server is found, no broadcasts are sent. The LDAP search is very fast, so if it fails, there is no discernible delay.

The Server Enumeration utility (dblocate) also uses LDAP—all database servers listed in LDAP are added to the list of database servers returned. This allows the Server Enumeration utility (dblocate) to list database servers that wouldn't be returned normally, for example, those which broadcasts wouldn't reach. Entries with timestamps older than 10 minutes are not included.

Example

The following is a sample *saldap.ini* file:

```
[LDAP]
server=ldapserver
basedn=dc=iAnywhere,dc=com
authdn=cn=SAServer,ou=iAnywhereASA,dc=iAnywhere,dc=com
password=secret
```

The entries are stored in a subtree of the basedn called iAnywhereASA. This entry must be created before SQL Anywhere can use LDAP. To create the subtree, you can use the LDAPADD utility, supplying the following information:

```
dn: ou=iAnywhereASA,basedn
objectClass: organizationalUnit
objectClass: top
ou: iAnywhereASA
```

Adjusting communication compression settings to improve performance

Enabling compression for one or all connections, and setting the minimum size at which packets are compressed, can improve SQL Anywhere performance in some circumstances.

To determine if enabling compression will help in your particular situation, it is recommended that you conduct a performance analysis on your particular network and using your particular application before using communication compression in a production environment. Performance results will vary according to the type of network you are using, your applications, and the data you transfer.

The most basic way of tuning compression is as simple as enabling or disabling the Compression (COMP) connection parameter on either the connection or server level. More advanced fine tuning of compression performance is available by adjusting the CompressionThreshold (COMPTH) connection parameter.

Enabling compression increases the quantity of information stored in data packets, thereby reducing the number of packets required to transmit a particular set of data. By reducing the number of packets, the data can be transmitted more quickly.

For more information about performance analysis, see “Performance Monitor statistics” [*SQL Anywhere Server - SQL Usage*], and “sa_conn_compression_info system procedure” [*SQL Anywhere Server - SQL Reference*].

Enabling compression

Enabling compression for a connection (or all connections) can significantly improve SQL Anywhere performance under some circumstances, including:

- When used over slow networks such as some wireless networks, some modems, serial links and some WANs.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- When used in conjunction with SQL Anywhere encryption over a slow network with built-in compression, since packets are compressed before they are encrypted.

Enabling compression, however, can sometimes also cause slower performance. For instance:

- Communication compression uses more memory and more CPU. It may cause slower performance, especially for LANs and other fast networks.

- Most modems and some slow networks already have built-in compression. In these cases, SQL Anywhere communication compression will not likely provide additional performance benefits unless you are also encrypting the data.

For more information about compression, see "Compress (COMP) connection parameter" on page 276 and "-pc dbsrv12 server option" on page 212.

Modifying the compression threshold

You can also adjust the compression threshold to improve SQL Anywhere performance. For most networks, the compression threshold does not need to be changed.

When compression is enabled, individual packets may or may not be compressed, depending on their size. For example, SQL Anywhere does not compress packets smaller than the compression threshold, even if communication compression is enabled. As well, small packets (less than about 100 bytes) usually do not compress at all. Since CPU time is required to compress packets, attempting to compress small packets could actually decrease performance.

Generally speaking, lowering the compression threshold value may improve performance on very slow networks, while raising the compression threshold may improve performance by reducing CPU usage. However, since lowering the compression threshold value will increase CPU usage on both the client and server, a performance analysis should be done to determine whether changing the compression threshold is beneficial.

See "CompressionThreshold (COMPTH) connection parameter" on page 277, and "-pt dbsrv12 server option" on page 213.

To adjust SQL Anywhere compression settings

1. Enable communication compression.

   Large data transfers with highly compressible data and larger packet sizes tend to get the best compression rates.

   For more information about enabling compression, see "Compress (COMP) connection parameter" on page 276, and "-pc dbsrv12 server option" on page 212.

2. Adjust the CompressionThreshold setting.

   Lowering the compression threshold value may improve performance on very slow networks, while raising the compression threshold may improve performance by reducing CPU usage.

For more information about adjusting the CompressionThreshold (COMPTH) connection parameter, see "CompressionThreshold (COMPTH) connection parameter" on page 277, and "-pt dbsrv12 server option" on page 213.

SQL Anywhere database connections

A database connection forms a channel through which all activity from the client application takes place. Client applications cannot interact with the database server until a connection is made. When the database server connection is made, a user's permissions determine what actions the user is authorized to perform on the database server.

When a user connects to a database, the database server assigns the user's connection a unique **connection ID**. For each new connection to the database server, the server increments the connection ID value by 1. These connection IDs are logged in the database server message log, which records informational messages, errors, warnings, and messages from the MESSAGE statement. The connection ID can be used to perform the following tasks:

- filter request logging information
- identify which connection has a lock on the database
- track the total number of connections to a server since it started and the order in which those connections were made

You can use the CONNECTION_PROPERTY function to obtain a user's connection ID by querying the Number connection property. See "Number connection property" on page 632.

See also

- "Logging database server actions" on page 41
- "Request logging" [*SQL Anywhere Server - SQL Usage*]
- "How locking works" [*SQL Anywhere Server - SQL Usage*]
- "-z dbeng12/dbsrv12 server option" on page 243

Using connection parameters

When connecting to a database, the application uses a set of **connection parameters** to define the connection. Connection parameters specify information such as the database server name, the database name, and the user ID. For a list of all supported connection parameters, see "Connection parameters" on page 265.

Each connection parameter specifies a keyword-value pair of the form *parameter=value*. The following example specifies the password connection parameter for the default password:

```
Password=sql
```

Connection parameters are assembled into **connection strings**. In a connection string, a semicolon separates each connection parameter:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
Host=demo12;DatabaseName=demo;ServerName=myserver
```

Representing connection strings

In the SQL Anywhere documentation, connection string examples can be represented in the following form:

```
parameter1=value1
parameter2=value2
...
```

This format is equivalent to the following connection string:

```
parameter1=value1;parameter2=value2
```

In your application, you **must** enter a connection string with the parameter settings separated by semicolons.

Connecting to SQL Anywhere databases

A single computer can run multiple database servers at the same time. Each database server can run multiple databases at the same time. The connection parameters required to connect to a database vary depending on whether multiple database servers are running on the same computer, whether multiple databases are running on the same database server, and whether the application is running on the same computer as the database server.

The following table describes several common connection scenarios and the client connection parameters that are required in each case. By combining these connection parameters, you can correctly identify the database you want to connect to. SQL Anywhere supports many connection parameters in addition to the ones listed below to handle less common connection scenarios.

Scenario	Required connection parameters	Comments	See also
One database running on a single database server, all running on the same computer as the client application.	Userid (UID) Password (PWD)		“Userid (UID) connection parameter” on page 310 “Password (PWD) connection parameter” on page 302
Database server running on a different computer than the client application.	Userid (UID) Password (PWD) HOST	The HOST connection parameter identifies the computer where the database server is running.	“Host connection parameter” on page 291

Scenario	Required connection parameters	Comments	See also
Multiple database servers running on the same computer.	Userid (UID) Password (PWD) ServerName (Server)	The Server connection parameter identifies the database server where the database is running. It is a good practice to always specify a server name.	"ServerName (Server) connection parameter" on page 306
Multiple databases running on the same database server that is only database server running on the computer.	Userid (UID) Password (PWD) DatabaseName (DBN)	The DBN connection parameter identifies which database you want to connect to.	"DatabaseName (DBN) connection parameter" on page 282
You do not know if the database is running on the database server and you want the database started so you can connect to it.	Userid (UID) Password (PWD) DatabaseFile (DBF)	The DBF connection parameter specifies the full path and name of the database you want to connect to, relative to the database server computer. The DBF connection parameter should specify a file on the computer where the database server is running, and it should not be a UNC file name.	"DatabaseFile (DBF) connection parameter" on page 280

See also

- "Connection parameters" on page 265

Connection parameters passed as connection strings

Connection parameters are passed to the interface library as a **connection string**. This string consists of a set of parameters, separated by semicolons:

```
parameter1=value1;parameter2=value2;...
```

Generally, the connection string that is built by an application and passed to the interface library does not correspond directly to the way users enter information. Instead, a user may complete a window, or the application may read connection information from an initialization file.

Many of the SQL Anywhere utilities accept a connection string as the -c option and pass the connection string unchanged to the interface library. The following example is a typical Backup utility (dbbackup) command line:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
dbbackup -c "HOST=myhost;DBN=demo;UID=DBA;PWD=sql" SQLAnybackup
```

See also

- "Connection parameters" on page 265
- "Network protocol options" on page 311
- "Character set conversion" on page 410

Connection parameter syntax rules

- **Connection strings containing spaces** You must enclose the entire connection string in double quotes if any of the connection parameter values contain spaces.
- **Boolean values** Boolean (true or false) arguments are either YES, ON, 1, TRUE, Y, or T if true, or NO, OFF, 0, FALSE, N, or F if false.
- **Case sensitivity** Connection parameters are case insensitive, although their values may not be (for example, file names on Unix).

  In order of precedence, you can be get the connection parameters used by the interface library from the following places:

  - **Connection string** You can pass parameters explicitly in the connection string.
  - **SQLCONNECT environment variable** The SQLCONNECT environment variable can store connection parameters.
  - **Data sources** ODBC data sources can store parameters.
- **Character set restrictions** It is recommended that the database server name (specified by the ServerName or SERVER connection parameter) be composed of the ASCII character set in the range 1 to 127. This limitation does not apply to other connection parameter values.
- **Priority** The following rules govern the priority of parameters:

  - The entries in a connect string are read left to right. If the same parameter is specified more than once, the last one in the string applies. ODBC, OLE DB, Sybase Central, Interactive SQL, and the SQL Anywhere Console utility are exceptions to this: if the same parameter is specified more than once, the first string applies.
  - If a string contains a data source or file data source entry, the profile is read from the configuration file, and the entries from the file are used if they are not already set. For example, if a connection string contains a data source name and sets some of the parameters contained in the data source explicitly, then the explicit parameters are used if a conflict occurs.
- **Connection string parsing** If there is a problem parsing the connection string, an error is generated that indicates which connection parameter caused the problem.

- **Empty connection parameters** Connection parameters that are specified with empty values are treated as a zero length string.

See also

- "Connection parameters" on page 265
- "Connection strings and character sets" on page 410

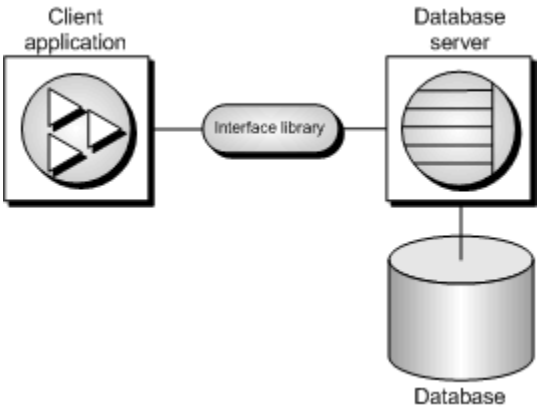
Connecting with SQL Anywhere APIs

To connect to a database, the client application must call one of the following SQL Anywhere API functions:

Interface	Details
ODBC	"ODBC support" [*SQL Anywhere Server - Programming*] "Creating ODBC data sources" on page 98
OLE DB	"Connecting to a database using OLE DB" on page 106
ADO.NET	"Connecting to a database" [*SQL Anywhere Server - Programming*]
Embedded SQL	"Embedded SQL" [*SQL Anywhere Server - Programming*]
Sybase Open Client	"Using SQL Anywhere as an Open Server" on page 1165 "Sybase Open Client support" [*SQL Anywhere Server - Programming*]
SQL Anywhere JDBC driver	"Connecting from a JDBC client application" [*SQL Anywhere Server - Programming*] "JDBC support" [*SQL Anywhere Server - Programming*]
jConnect JDBC driver	"Connecting from a JDBC client application" [*SQL Anywhere Server - Programming*] "JDBC support" [*SQL Anywhere Server - Programming*]

The SQL Anywhere API uses connection information included in the call from the client application to locate and connect to the database server. Information sent by the client application can include information held in a data source, the SQLCONNECT environment variable, or the server address cache. The following figure is a simplified representation of the process.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0



Additional information

If you want ...	Consider reading ...
An overview of connecting from Sybase Central, Interactive SQL, or the SQL Anywhere Console utility	"Working with the Connect window" on page 92
Some examples to get started quickly, including Sybase Central and Interactive SQL scenarios	"Sample SQL Anywhere database connections" on page 127
To learn about data sources	"Creating ODBC data sources" on page 98
To learn what connection parameters are available	"Connection parameters" on page 265
To see an in-depth description of how connections are established	"Troubleshooting connections" on page 138
To learn about network-specific connection issues	"Client/server communications" on page 76
To learn about character set issues affecting connections	"Connection strings and character sets" on page 410
To learn about connecting through a firewall	"Connecting across a firewall" on page 80

Connecting from desktop applications to a Windows Mobile database

You can connect from applications running on a desktop computer, such as Sybase Central or Interactive SQL, to a database server running on a Windows Mobile device. The connection uses TCP/IP over the Microsoft ActiveSync (Windows Mobile Device Center) link between the desktop computer and the Windows Mobile device.

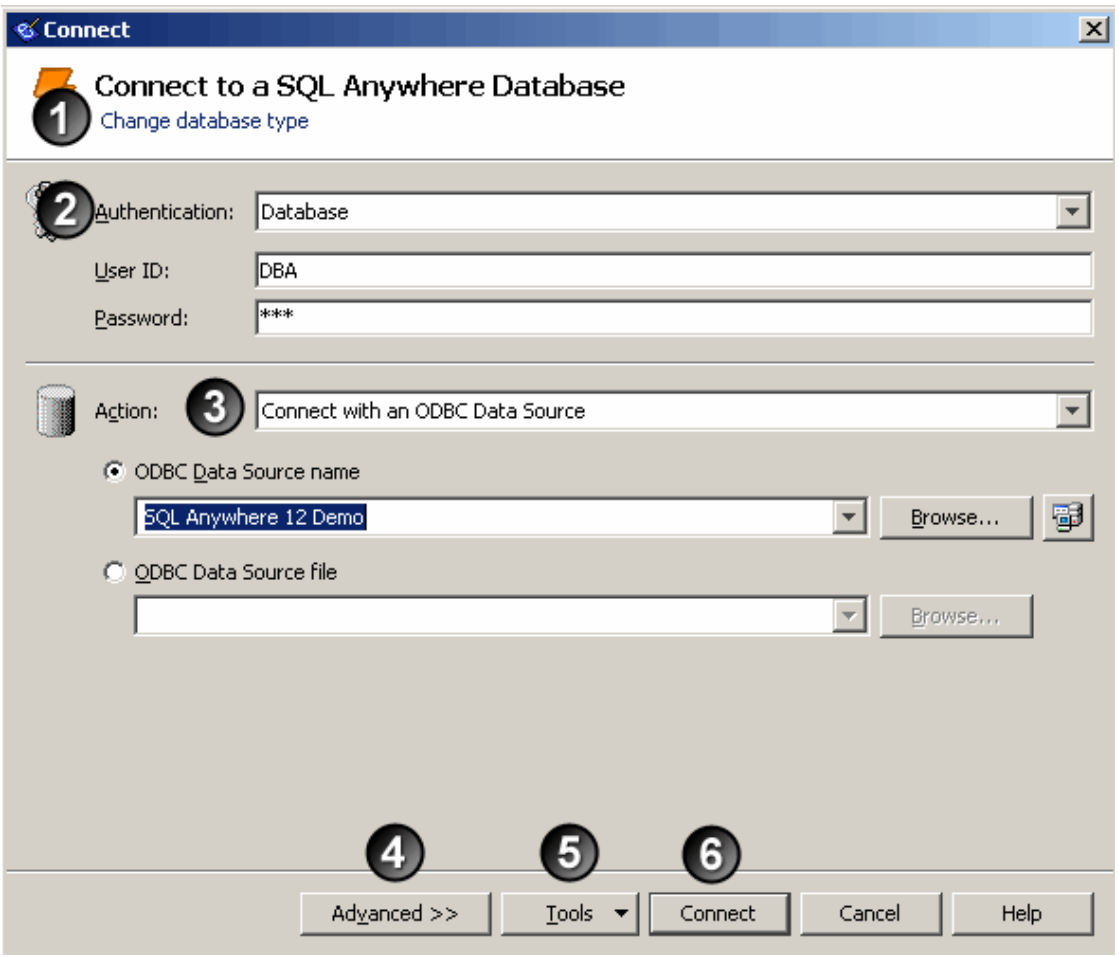
See also

- “Start a database server on your Windows Mobile device” on page 349
- “Create an ODBC data source to connect to your Windows Mobile device” on page 350
- “Determine the IP address of your Windows Mobile device” on page 352

Working with the Connect window

When connecting to a server or database from Sybase Central, Interactive SQL, or the SQL Anywhere Console utility, you use the **Connect** window to specify connection parameters. Information you enter in the **Connect** window is preserved between sessions.

The connection parameters you specify in the **Connect** window are dependent on the connection type.



 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To connect to a database using the Connect window

1. If required, click **Change Database Type** and choose a database type. For example, to connect to a SQL Anywhere database, select **SQL Anywhere**.
2. In the **Authentication** dropdown list, choose either **Database** to connect using your user ID and password or **Windows Integrated Login** to connect using your Windows Integrated Login.

   If you choose **Database**, then:

   a. In the **User ID** field, type a user name. For example, type **DBA**.

   b. In the **Password** field, type a password for the database. For example, type **sql**.
3. From the **Action** dropdown list, choose a connection type:

   Connect To A Running Database On This Computer Connects to a database that is already running on your computer.

   Connect With An ODBC Data Source Connect to a database using an ODBC data source.

   Connect To A Running Database On Another Computer Connects to a database that is running on another computer in the network.

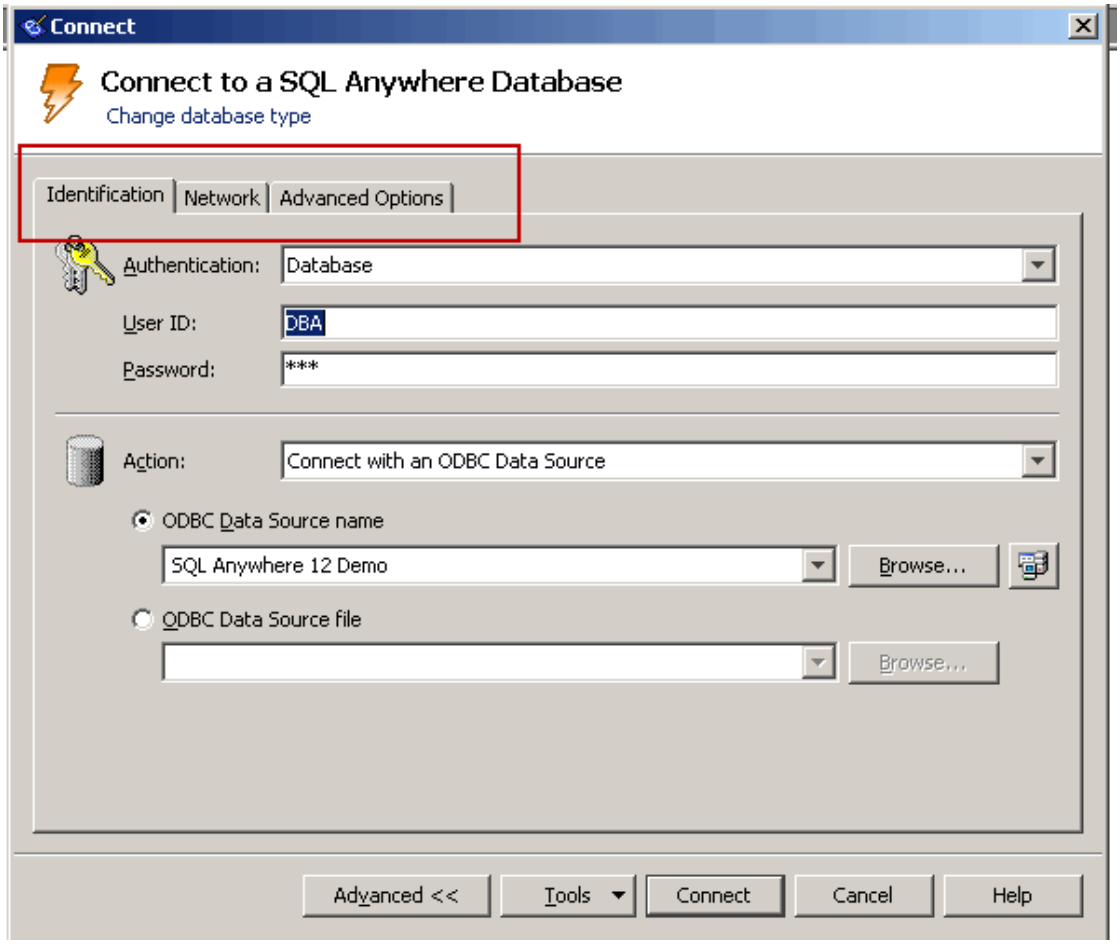
   Start And Connect To A Database On This Computer Starts a database on this computer and connects to it.

   Start And Connect To A Database On Another Computer Starts a database over a network on another computer and connects to it.

   Connect With A Connection String Connects to a database using a connection string.

   The connection parameters below the **Action** dropdown list , which are also chosen in this step change depending upon your choice.
4. If required, you can click **Advanced** to specify options such as TCP/IP, encryption, and other advanced options.

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0



5. If required, click **Tools**, to access the following tools:

   Test Connection This tool tests whether the information provided results in a proper connection. This tool is only available when you are connecting to a SQL Anywhere database.

   Copy Connection String to Clipboard This tool copies the connection string, which is based on the information provided to the clipboard.

   Save as ODBC Data Source This tool creates an ODBC data source based on the information provided. See "Create ODBC data sources using the Connect window" on page 99.

   Setup Windows Mobile proxy port This tool helps you set up a TCP/IP port redirector for connecting to Windows Mobile databases (if Windows Mobile services are installed). This tool is only available if you have Microsoft ActiveSync installed and are connecting to a SQL Anywhere or UltraLite database.

6. Click **Connect**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Sample SQL Anywhere database connections" on page 127
- "Starting the SQL Anywhere Console utility" on page 760

Open the Connect window

If the **Connect** window does not open automatically, use one of the following procedures to open it.

To open the Connect window (Sybase Central)

1. Start Sybase Central. See "Starting Sybase Central" on page 680.
2. Choose **Connections** » **Connect With SQL Anywhere 12**.

   Alternatively, you can press F11 to open the **Connections** menu.

Tip
You can make subsequent connections to a given database easier and faster using a **connection profile**. See "Sybase Central connection profiles" on page 95.

To open the Connect window (Interactive SQL)

1. Start Interactive SQL. See "Starting Interactive SQL" on page 698.
2. Choose **SQL** » **Connect**.

   Alternatively, you can press F11 to open the **Connect** window.

To open the Connect window (SQL Anywhere Console utility)

1. Start the SQL Anywhere Console utility. See "Starting the SQL Anywhere Console utility" on page 760.
2. Choose **Command** » **Connect**.

Sybase Central connection profiles

When you first connect to a database server or database, you enter a user ID, password, and other connection parameters. This information must be entered again when you make subsequent connections. To save time and simplify the connection process, you can create a connection profile to save the connection parameters for each database.

To use and manage connection profiles, choose **Connections** » **Connection Profiles**. This command opens the **Connection Profiles** window, where you can:

- connect using a connection profile
- edit an existing connection profile
- create a new connection profile
- set a description for a profile
- delete or remove profiles
- import or export a connection profile
- set a profile to connect automatically when Sybase Central is started

Note
Connection profiles are specific to Sybase Central. If you are building an ODBC application, you can use ODBC data sources to achieve functionality similar to connection profiles. See "Creating ODBC data sources" on page 98.

Create a connection profile

To create a new connection profile (Sybase Central)

1. Choose **Connections** » **Connection Profiles**.
2. Click **New**.
3. In the **Name** field, type a name for the new profile.
4. Select **New Connection Profile** and choose the appropriate plug-in from the list. The plug-in is the product, such as **SQL Anywhere 12** or **SQL Anywhere 12**.

   To base your new connection profile on an existing profile, select **Copy Connection Profile** and choose the profile from the **Existing Connection Profiles** list.
5. To allow other users to access the profile, select **Share This Connection Profile With Other Users**. This setting is useful on multi-user platforms such as Unix.
6. Click **OK**.
7. In the **Edit Connection Profile** window, enter the required values, and then click **OK** to close the window.

Connect automatically using a connection profile

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To connect automatically when Sybase Central starts

1. Choose **Connections** » **Connection Profiles**.
2. In the **Connection Profiles** list, select a connection profile.
3. Click **Set Startup** to change the **Use On Startup** column from **No** to **Yes**.

Edit a connection profile

To edit the parameters of an existing connection profile (Sybase Central)

1. Choose **Connections** » **Connection Profiles**.
2. In the **Connection Profiles** list, select a connection profile.
3. Click **Edit**.
4. In the **Edit Connection Profile** window, edit the values.

Import a connection profile

To import a connection profile (Sybase Central)

1. Choose **Connections** » **Connection Profiles**.
2. Click **Import**.
3. In the **File Name** field, type the name of the connection profile file you want to import.
4. Click **OK**.

Export a connection profile

To export a connection profile (Sybase Central)

1. Choose **Connections** » **Connection Profiles**.
2. In the **Connection Profiles** list, select a connection profile.
3. Click **Export**.
4. In the **File Name** field, type a file name for the connection profile.
5. Click **Save**.

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Creating ODBC data sources

Microsoft **Open Database Connectivity** (**ODBC**) is a standard application programming interface for connecting client applications to Windows-based database management systems.

Many client applications, including application development systems, use the ODBC interface to access SQL Anywhere. When connecting to the database, ODBC applications typically use ODBC data sources. An ODBC data source is a set of connection parameters, stored in the registry or in a file.

Caution
Storing user IDs, encrypted or unencrypted passwords, and database keys in a data source is not recommended.

The SQL Anywhere ODBC driver is named *dbodbc12.dll*, and it is located in *install-dir\bin32*.

For more information about using SQL Anywhere with ODBC, see "ODBC conformance" [*SQL Anywhere Server - Programming*].

You can use ODBC data sources to connect to SQL Anywhere databases from the following applications:

- Sybase Central, Interactive SQL, and the SQL Anywhere Console utility.
- All SQL Anywhere utilities.
- Sybase PowerDesigner Physical Data Model and InfoMaker.
- Any application development environment that supports ODBC, such as Microsoft Visual Basic, Sybase PowerBuilder, and Borland Delphi.
- SQL Anywhere client applications on Unix. On Unix, the data source is stored as a file.

Storing SQL Anywhere connection parameters

You use an ODBC data source to connect to an ODBC database. The client computer requires an ODBC data source for each database connection.

The ODBC data source contains a set of connection parameters. You can store sets of SQL Anywhere connection parameters as an ODBC data source, in either the Windows registry or as files.

For SQL Anywhere, the use of ODBC data sources goes beyond Windows applications using the ODBC interface:

- SQL Anywhere client applications on Unix and Windows operating systems can use ODBC data sources.
- ODBC data sources can be used by all SQL Anywhere client interfaces except jConnect and Sybase Open Client. The data source is stored in a file on Unix and Windows Mobile operating systems.

If you have a data source, your connection string can name the data source to use:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Data source** Use the DataSourceName (DSN) connection parameter to reference a data source in the Windows registry:

  ```
  DSN=my-data-source
  ```

- **File data source** Use the FileDataSourceName (FILEDSN) connection parameter to reference a data source held in a file:

  ```
  FileDSN=mysource.dsn
  ```

Note
When creating a connection string, it can contain the name of an ODBC data source that contains connection parameters, and connection parameters that are specified explicitly. If a connection parameter is specified in the connection string and in the ODBC data source, the value that is specified in the connection string takes precedence.

See also

- "Connection parameters" on page 265

Create ODBC data sources using the Connect window

Use the **Connect** window to create ODBC data sources in Sybase Central, Interactive SQL, and the SQL Anywhere Console utility.

To create an ODBC data source using the Connect window

1. Open the **Connect** window. See "Open the Connect window" on page 95.
2. If required, click **Change Database Type**, and then choose **SQL Anywhere**.
3. Specify a **User ID**, **Password**.
4. From the **Action** dropdown list, choose **Connect To A Database On This Computer** to connect to a database that is running on your computer.
5. Specify a **Database File Name**.
6. Choose **Tools** » **Save As ODBC Data Source**.
7. In the **Enter the name for this new data source** field, type a name for the data source.
8. In the **Select The Data Source Type** list, specify whether the data source is available for the current user or all users.
9. Click **Save**.
10. Click **Connect**.

See also

- "Connect using a data source" on page 132

Create ODBC data sources using the ODBC Data Source Administrator

Use the Microsoft ODBC Data Source Administrator to create and edit data sources on Windows-based applications. Use the utility to work with User Data Sources, File Data Sources, and System Data Sources.

Caution
Storing user IDs, encrypted or unencrypted passwords, and database keys in a data source is not recommended.

To create an ODBC data source (ODBC Data Source Administrator)

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **ODBC Data Source Administrator**.
2. To create an ODBC data source for the current user, click the **User DSN** tab.

   To create a system-wide ODBC data source, click the **System DSN** tab.
3. Click **Add**.
4. In the **Name** list, choose **SQL Anywhere 12**. Click **Finish**.
5. Specify the connection parameters for the ODBC data source.
6. Click **OK**.
7. Click **OK**.

Creating a System ODBC data source on 64-bit Windows

64-bit versions of Windows maintain two sets of the System Data Source collection; one for 64-bit applications and one for 32-bit applications. To create a System Data Source that is accessible to both 64-bit and 32-bit applications, you must run a copy of the 32-bit ODBC Data Source Administrator (located in the *WINDOWS\SysWOW64* folder). To avoid connection problems, set up your 32-bit System Data Source exactly like your 64-bit System Data Source.

To edit an ODBC data source using the ODBC Data Source Administrator

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **ODBC Data Source Administrator**.
2. Click one of the **User DSN** or **System DSN** tabs.
3. In the **Name** list, click a data source.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

4. Click **Configure**.
5. Edit the connection parameters for the ODBC data source.
6. Click **OK**.
7. Click **OK**.

See also

- "Connect using a data source" on page 132

Create an ODBC data source with the dbdsn utility

File Data Sources can not be created with the dbdsn utility. Use the ODBC Data Source Administrator to create File Data Sources. System Data Sources are limited to Windows-based operating systems.

> **Caution**
> Storing user IDs, encrypted or unencrypted passwords, and database keys in a data source is not recommended.

To create an ODBC data source (command line)

- Run a dbdsn command, specifying the connection parameters you want to use.

  For example, the following command creates a data source for the sample database. The command must be entered on one line:

  ```
  dbdsn -w "My DSN" -c "DBF=samples-dir\demo.db"
  ```

  For information about *samples-dir*, see "Samples directory" on page 392.

For more information about the dbdsn utility, see "Data Source utility (dbdsn)" on page 779.

Creating a System ODBC data source on 64-bit Windows

64-bit versions of Windows maintain two sets of the System Data Source collection; one for 64-bit applications and one for 32-bit applications. To create a System Data Source that is accessible to both 64-bit and 32-bit applications, you must run the 32-bit version of dbdsn (located in the SQL Anywhere *bin32* folder). To avoid connection problems, set up your 32-bit System Data Source exactly like your 64-bit System Data Source.

See also

- "Connect using a data source" on page 132

Create an ODBC data source on Mac OS X

The SQL Anywhere ODBC driver must be added before you create the ODBC data source.

Caution
Storing user IDs, encrypted or unencrypted passwords, and database keys in a data source is not recommended.

To add the SQL Anywhere ODBC driver

1. If necessary, go to http://mac.softpedia.com/progDownload/ODBC-Administrator-Tool-Download-61963.html to download the ODBC Administrator tool.

2. Launch the ODBC Data Source Administrator from */Applications/Utilities*.

3. Select the **Drivers** tab.

4. Click **Add**.

5. In the **Description** field, type **SQL Anywhere 12**.

6. Click **Choose** and select the SQL Anywhere ODBC driver in both the **Driver File Name** and **Setup File Name** fields. By default, it is located in */Applications/SQLAnywhere12/System/lib32/dbodbc12_r.bundle*.

   The *_r* in the bundle name indicates that it is the threaded version of the driver. There is also an unthreaded version (*dbodbc12.bundle*) for use with unthreaded applications.

7. Click **OK**.

You can add the information with a text editor. The ODBC configuration files are located in */Library/ODBC* within your home directory. There is an *odbcinst.ini* file for driver information and an *odbc.ini* file for data source information.

You can also use the Data Source utility (dbdsn) to create ODBC data sources on Mac OS X. See "Data Source utility (dbdsn)" on page 779.

To create an ODBC data source

1. Launch the ODBC Data Source Administrator from */Applications/Utilities*.

2. In the ODBC Data Source Administrator, click the **User DSN** tab, and then click **Add**.

3. In the **Name** list, click **SQL Anywhere 12**.

4. Click **OK**.

5. In the **Data Source Name** field, type **Demo12**.

6. Add the following connection parameters. The connection parameters and values are case insensitive.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Keyword	Value
UserID	DBA
Password	sql
StartLine	dbeng12
DatabaseFile	*/Applications/SQLAnywhere12/System/demo.db*
ThreadManager	ON
Driver	SQL Anywhere 12

For more information about connection parameters, see "Connection parameters" on page 265.

7. Click **OK**.
8. Click **Apply**.
9. Press Command+Q to exit the ODBC Data Source Administrator.

See also

- "Connect using a data source" on page 132
- "Using ODBC data sources on Unix" on page 105

Using file data sources on Windows

Generally, on Windows-based operating systems, ODBC data sources are stored in the system registry. File data sources are an alternative, which are stored as files. In Windows, file data sources typically have the extension *.dsn*. They consist of sections, each section starting with a name enclosed in square brackets.

To connect using a File Data Source, use the FileDataSourceName (FILEDSN) connection parameter. You can not use both DataSourceName (DSN) and FileDataSourceName (FILEDSN) in the same connection.

To successfully create a File Data Source, you must be able to establish a connection to the database for which you are creating a File Data Source.

If the connection is not successful, one of two things will happen. For a new File Data Source, the Microsoft ODBC Data Source Administrator will display the message **"A connection could not be made using the file data source parameters entered. Save non-verified file DSN?"**. If you choose to save the File Data Source, the ODBC Data Source Administrator writes only the following lines to the file.

```
[ODBC]
DRIVER=SQL Anywhere 12
```

The driver string varies with the version of SQL Anywhere. For an existing File Data Source, the ODBC Data Source Administrator displays the message `General error: Invalid file dsn 'file-data-source.dsn'`. The File Data Source is not updated by the ODBC Data Source Administrator.

If the connection is successful, the new or updated File Data Source is written to disk by the ODBC Data Source Administrator, but the PWD=password parameter is not included (and is removed if it was previously present in the file).

Note
Use File Data Sources to distribute the file to users and simplify the management of multiple user connections. If the file is placed in the default location for file data sources, it is picked up automatically by ODBC.

To create an ODBC file data source (ODBC Data Source Administrator)

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **ODBC Data Source Administrator**.
2. Click the **File DSN** tab.
3. Click **Add**.
4. In the **Name** list, click **SQL Anywhere 12**.
5. Click **Next**.
6. Follow the instructions in the **Create New Data Source Wizard**.
7. Click **OK**.
8. Click **OK**.

Using ODBC data sources on Windows Mobile

Windows Mobile does not provide an ODBC driver manager or an ODBC data source administrator. On Windows Mobile, SQL Anywhere uses ODBC data sources stored in files. To use these data source definitions, use either the DSN or the FILEDSN connection parameter; on Windows Mobile DSN and FILEDSN are synonyms.

Caution
Storing user IDs, encrypted or unencrypted passwords, and database keys in a data source is not recommended.

Data source location

Windows Mobile searches for the data source files in the root directory of the device: *\filename.dsn*.

Each data source is held in a file. The file has the same name as the data source, with an extension of *.dsn*.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

A sample Windows Mobile data source

The following is a sample of an ODBC data source for Windows Mobile.

```
[ODBC]
DRIVER=\windows\dbodbc12.dll
Integrated=No
AutoStop=Yes
Host=192.168.0.55
LOG=\sa_connection.txt
START=dbsrv12 -c 8M
SERVER=demo
```

See also

- "Create an ODBC data source to connect to your Windows Mobile device" on page 350
- "Using file data sources on Windows" on page 103

Using ODBC data sources on Unix

On Unix operating systems, ODBC data sources are held in a system information file. This file is usually named *.odbc.ini*. The SQL Anywhere database server searches the following locations, in order, for the system information file:

- The ODBCINI environment variable.
- The ODBC_INI environment variable.
- The ODBCHOME environment variable.
- The HOME environment variable.
- The user's home directory (~).
- The PATH environment variable.

Note
The ODBCINI and ODBC_INI environment variables can be used to locate the system information file (which is usually named *.odbc.ini*), while the ODBCHOME and HOME environment variables can be used to define the path where the *.odbc.ini* file is located.

Both ODBCINI and ODBC_INI specify a full path, including the file name. If the system information file is located in a directory specified by ODBCINI or ODBC_INI, it does not have to be named *.odbc.ini*.

The following is a sample system information file:

```
[My Data Source]
Host=hostname
```

You can enter any connection parameter in the system information file. See "Connection parameters" on page 265.

Network protocol options are added as part of the CommLinks (LINKS) parameter. See "Network protocol options" on page 311.

Caution
Storing user IDs, encrypted or unencrypted passwords, and database keys in a data source is not recommended.

On Unix, use the dbdsn utility to create and manage ODBC data sources.

Caution
On Unix, do not add simple encryption to the system information file (named *.odbc.ini* by default) with the File Hiding utility (dbfhide) unless you are using only SQL Anywhere data sources. If you plan to use other data sources (for example, for MobiLink synchronization), then obfuscating the contents of the system information file may prevent other drivers from functioning properly.

See also

- "Creating ODBC data sources" on page 98
- "Data Source utility (dbdsn)" on page 779
- "ODBCHOME environment variable [Unix]" on page 380
- "ODBCINI and ODBC_INI environment variables [Unix]" on page 380

Connecting to a database using OLE DB

This section describes how to connect to a SQL Anywhere database using OLE DB in the following environments:

- Microsoft ActiveX Data Objects (ADO) provides a programming interface for OLE DB data sources. You can access SQL Anywhere from programming tools such as Microsoft Visual Basic.
- Sybase PowerBuilder can access OLE DB data sources, and you can use SQL Anywhere as a Sybase PowerBuilder OLE DB database profile.

OLE DB uses the Component Object Model (COM) to make data from a variety of sources available to applications. Relational databases are among the classes of data sources that you can access through OLE DB.

See also

- "Introduction to OLE DB" [*SQL Anywhere Server - Programming*]

OLE DB providers

An OLE DB provider is required for each type of data source you want to access. Each **OLE DB provider** is a dynamic-link library. To access SQL Anywhere, choose one of the following OLE DB providers:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **SQL Anywhere OLE DB Provider** The SQL Anywhere OLE DB Provider provides access to SQL Anywhere as an OLE DB data source without the need for ODBC components. The short name for this provider is **SAOLEDB**.

  The SAOLEDB provider is self registering. This registration process includes making registry entries in the COM section of the registry so that ADO can locate the DLL when the SAOLEDB provider is called. If you change the location of your DLL, you must re-register it.

- **Microsoft OLE DB provider for ODBC** Microsoft provides an OLE DB provider with a short name of **MSDASQL**.

  The MSDASQL provider makes ODBC data sources appear as OLE DB data sources. It requires the SQL Anywhere ODBC driver.

See also

- "Introduction to OLE DB" [*SQL Anywhere Server - Programming*]

Connecting from ADO

ADO is an object-oriented programming interface. In ADO, the **Connection** object represents a unique session with a data source.

You can use the following Connection object features to initiate a connection:

- The Provider property that holds the name of the provider. If you do not supply a Provider name, ADO uses the MSDASQL provider.
- The ConnectionString property that holds a connection string. This property holds a SQL Anywhere connection string, which is used in the same way as the ODBC driver. You can supply ODBC data source names, or explicit UserID, Password, DatabaseName, and other parameters, just as in other connection strings.
- The Open method initiates a connection.

Example

The following Microsoft Visual Basic code initiates an OLE DB connection to SQL Anywhere:

```
' Declare the connection object
Dim myConn as New ADODB.Connection
myConn.Provider = "SAOLEDB"
myConn.ConnectionString = "DSN=SQL Anywhere 12 Demo"
myConn.Open
```

See also

- "ADO programming with SQL Anywhere" [*SQL Anywhere Server - Programming*]

Using Windows integrated logins

The Windows **integrated login** feature allows you to maintain a single user ID and password for operating system and network logins, and database connections. To create an integrated login:

- Enable the integrated login feature.
- Create a database user to map the integrated login to (if one does not already exist).
- Create an integrated login mapping between a Windows user or group profile and an existing database user. The **Login Mappings** folder in Sybase Central lists all users with integrated login permissions.
- Connect from a client application and test the integrated login facility.

Supported operating systems

Integrated login capabilities are available for Windows-based database servers. Windows clients can use integrated logins to connect to a network server running on Windows.

Integrated login benefits

An integrated login is a mapping from one or more Windows users or Windows user group profiles to an existing database user. A user who has successfully navigated the security for that user profile or group and logged in to a computer can connect to a database without providing an additional user ID or password.

To do this, the database must be configured to use integrated logins and a mapping must have been granted between the user or group profile used to log in to the computer or network, and a database user.

Using an integrated login is more convenient for the user and permits a single security system for database and network security. The advantages of an integrated login include:

- Users do not need to type a user ID or password.
- Users are authenticated by the operating system. A single system is used for database security and computer or network security.
- Multiple user or group profiles can be mapped to a single database user ID.
- The name and password used to login to the Windows computer do not have to match the database user ID and password.

Caution
Integrated logins offer the convenience of a single security system, but there are important security implications that database administrators should be familiar with. See “Security concerns: Unrestricted database access” on page 115 and “Security concerns: Copied database files” on page 126.

Enable the integrated login feature

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The login_mode database option determines whether the integrated login feature is enabled. As database options apply only to the database in which they are found, different databases can have a different integrated login setting even if they are loaded and running on the same server.

The login_mode database option accepts the following values:

- **Standard** Standard logins are permitted. This is the default setting. Standard connection logins must supply both a user ID and password, and do not use the Integrated or Kerberos connection parameters. An error occurs if an Integrated or Kerberos login connection is attempted.
- **Integrated** Integrated logins are permitted.
- **Kerberos** Kerberos logins are permitted. See "Kerberos authentication" on page 116.

Caution
Setting the login_mode database option to not allow Standard logins restricts connections to only those users or groups who have been granted an integrated or Kerberos login mapping. Attempting to connect with a user ID and password generates an error unless you are a user with DBA authority.

To allow more than one type of login, specify multiple values for the login_mode option. For example, the following SQL statement sets the value of the login_mode database option to allow both standard and integrated logins:

```
SET OPTION PUBLIC.login_mode = 'Standard,Integrated';
```

If a database file can be copied, the temporary public login_mode option should be used (both for integrated and Kerberos logins). This way, integrated and Kerberos logins are not supported by default if the file is copied.

Creating login mappings

SQL Anywhere lets you create login mappings to map a Windows user profile or Kerberos principal to an existing user in a database. You can create either type of login mapping by using the **Create Login Mapping Wizard** or the GRANT statement. You must have DBA authority to create login mappings. See "Create an integrated login" on page 109 and "Create Kerberos login mappings" on page 121.

See also

- "Using Windows integrated logins" on page 108
- "Kerberos authentication" on page 116

Create an integrated login

User profiles can only be mapped to an existing database user ID. When that database user ID is removed from the database, all integrated login mappings based on that database user ID are automatically removed.

A user or group profile does not have to exist for it to be mapped to a database user ID. More than one user profile can be mapped to the same database user ID.

You can use either the **Create Login Mapping Wizard** or a SQL statement to create an integrated login mapping.

You must have DBA authority to create or delete an integrated login mapping.

To map an integrated login (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, right-click **Login Mappings** and choose **New** » **Login Mapping**.
3. Click **Next**.
4. In the **Which Windows User Will Be Connecting To The Database** field, type the name of the user or group profile for whom the integrated login is to be created.
5. In the **Which Database User Do You Want To Associate With The Windows User** list, select the database user ID this user maps to.
6. Follow the remaining instructions in the **Create Login Mapping Wizard**.

To map an integrated login (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a GRANT INTEGRATED LOGIN TO statement.

Example

The following SQL statement allows Windows users fran_whitney and matthew_cobb to log in to the database as the user DBA, without having to know or provide the DBA user ID or password.

```
GRANT INTEGRATED LOGIN
TO fran_whitney, matthew_cobb
AS USER DBA;
```

See "GRANT statement" [*SQL Anywhere Server - SQL Reference*].

The following SQL statement allows Windows users who are members of the Windows NT group mywindowsusers to log in to the database as the user DBA, without having to know or provide the DBA user ID or password.

```
GRANT INTEGRATED LOGIN
TO mywindowsusers
AS USER DBA;
```

See "Creating integrated logins for Windows user groups" on page 112.

Revoke an integrated login permission

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To revoke an integrated login permission (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, click**Login Mappings**.
3. In the right pane, right-click the login mapping you want to remove and click **Delete**.
4. Click **Yes**.

To revoke an integrated login permission (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a REVOKE INTEGRATED LOGIN FROM statement.

Example

The following SQL statement removes integrated login permission from the Windows user pchin.

```
REVOKE INTEGRATED LOGIN
FROM pchin;
```

See "REVOKE statement" [*SQL Anywhere Server - SQL Reference*].

Connect to a database from a client application

To connect a client application to a database using an integrated login:

- Set the Integrated (INT) parameter in the list of connection parameters to YES.
- Do not specify a user ID or password in the connection string or **Connect** window.

If the Integrated (INT) parameter is set to YES in the connection string, an integrated login is attempted. The server attempts a standard login when the connection attempt fails and the login_mode database option is set to Standard,Integrated. See "login_mode option" on page 547.

If an attempt to connect to a database is made without providing a user ID or password, an integrated login is attempted. The success of the login attempt is dependent on whether the current user profile name matches an integrated login mapping in the database.

Interactive SQL examples

In the following example, the connection attempt succeeds when the user logs in with a user profile that matches the integrated login mapping in the default database server:

```
CONNECT USING 'INTEGRATED=yes';
```

The Interactive SQL statement CONNECT can connect to a database when:

- A server is currently running.

- The default database has the login_mode database option set to accept integrated login connections.
- An integrated login mapping has been created that matches the current user's user profile name or for a Windows user group to which the user belongs.
- A user clicks **OK** without providing more information when the more connection information prompt appears.

Creating integrated logins for Windows user groups

When a Windows user logs in, if they do not have an explicit integrated login mapping, but belong to a Windows user group for which there is an integrated login mapping, the user connects to the database as the database user or group specified in the Windows user group's integrated login mapping.

Caution
Creating an integrated login for a Windows user group allows any user that is a member of the group to connect to the database without knowing a user ID or password.

See "Prevent Windows user groups members from connecting to a database" on page 113.

Members of multiple groups

If the Windows user belongs to more than one Windows user group, and more than one Windows user group on the computer has an integrated login mapping in the database, then the integrated login only succeeds if all the Windows user groups on the computer have integrated login mappings to the same database user ID. If multiple Windows user groups have integrated login mappings to different database user IDs, an error is returned and the integrated login fails.

For example, consider a database with two user IDs, dbuserA and dbuserB, and the Windows user windowsuser who belongs to the Windows user groups xpgroupA and xpgroupB.

This SQL statement...	Allows...

`GRANT INTEGRATED LOGIN`
`TO windowsuser`
`AS USER dbuserA;`
	windowsuser to connect to the database using the integrated login mapping set explicitly for windowsuser.

`GRANT INTEGRATED LOGIN`
`TO xpgroupA`
`AS USER dbuserB;`
	windowsuser to connect to the database using the integrated login mapping granted to xpgroupA.

`GRANT INTEGRATED LOGIN`
`TO xpgroupA`
`AS USER dbuserB;`
`GRANT INTEGRATED LOGIN xpgroupb`
`AS USER dbuserB;`
	windowsuser to connect to the database because both Windows user groups that windowsuser belongs to have an integrated login mapping to the same database user.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

This SQL statement...	Allows...
`GRANT INTEGRATED LOGIN TO xpgroupA AS USER dbuserA; GRANT INTEGRATED LOGIN TO xpgroupb AS USER dbuserB;`	No connection to the database. When windowsuser attempts to connect to the database, the integrated login fails because each Windows user group has an integrated login mapping to a different database user and windowsuser is a member of both Windows user groups.

Domain Controller locations

By default, the computer on which the SQL Anywhere database server is running is used to verify Windows user group membership. If the Domain Controller server is on a different computer than the database server, you can specify the name of the Domain Controller server using the integrated_server_name option. For example:

```
SET OPTION PUBLIC.integrated_server_name = '\\myserver-1';
```

See "integrated_server_name option" on page 543.

Prevent Windows user groups members from connecting to a database

There are two methods you can use to prevent a user who is a member of a Windows user group with an integrated login from connecting to a database using the group integrated login:

- Create an integrated login for the user to a database user ID that does not have a password.
- Created a stored procedure that is called by the login_procedure option to check whether a user is allowed to log in, and raise an exception when a disallowed user tries to connect.

Creating an integrated login to a user ID with no password

When a user is a member of a Windows user group that has an integrated login, but also has an explicit integrated login for their user ID, the user's integrated login is used to connect to the database. To prevent a user from connecting to a database using their Windows user group integrated login, you can create an integrated login for the Windows user to a database user ID without a password. Database user IDs that do not have a password can not connect to a database.

To create an integrated login to a user ID with no password (SQL)

1. Add a user to the database without a password. For example:

   ```
   CREATE USER db_user_no_password;
   ```

2. Create an integrated login for the Windows user that maps to the database user without a password. For example:

```
GRANT INTEGRATED LOGIN TO WindowsUser
AS USER db_user_no_password;
```

Creating a procedure to prevent Windows users from connecting

The login_procedure option specifies the stored procedure to call each time a connection to the database is attempted. By default, the dbo.sp_login_environment procedure is called. You can set the login_procedure option to call a procedure you have written that prevents specific users from connecting to the database.

The following example creates a procedure named login_check that is called by the login_procedure option. The login_check procedure checks the supplied user name against a list of users that are not allowed to connect to the database. If the supplied user name is found in the list, the connection fails. In this example, users named Joe, Harry, or Martha are not allowed to connect. If the user is not found in the list, the database connection proceeds as usual and calls the sp_login_environment procedure.

```
CREATE PROCEDURE DBA.user_login_check()
   BEGIN
      DECLARE INVALID_LOGON EXCEPTION FOR SQLSTATE '28000';
      // Disallow certain users
      IF( CURRENT USER IN ('Joe','Harry','Martha') ) THEN
        SIGNAL INVALID_LOGON;
      ELSE
         CALL sp_login_environment;
      END IF;
   END
go
GRANT EXECUTE ON DBA.user_login_check TO PUBLIC
go
SET OPTION PUBLIC.login_procedure='DBA.user_login_check'
go
```

Network aspects of integrated logins

If the database is located on a network server, then one of two conditions must be met for integrated logins to be used:

- The user profile used for the integrated login connection attempt must be identical on both the local computer and the server. The passwords for both user profiles must also be identical.

  For example, when the user jsmith attempts to connect using an integrated login to a database loaded on a network server, identical user profile names and passwords must exist on both the local computer and the computer running the database server. The user jsmith must be permitted to log in to both computers.

- If network access is controlled by a Microsoft Domain, the user attempting an integrated login must have domain permissions with the Domain Controller server and be logged in to the network. A user profile on the network server matching the user profile on the local computer is not required.

Creating a default integrated login user

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

A default integrated login user ID can be created so that connecting via an integrated login will be successful even if no integrated login mapping exists for the user profile currently in use.

For example, if no integrated login mapping exists for the user profile name JSMITH, an integrated login connection attempt will normally fail when JSMITH is the user profile in use.

However, if you create a user ID named Guest in a database, an integrated login will successfully map to the Guest user ID if no integrated login mapping explicitly identifies the user profile JSMITH.

Caution
The default integrated login user permits anyone attempting an integrated login to connect to a database successfully if the database contains a user ID named Guest. The authorities granted to the Guest user ID determine the permissions and authorities granted to the newly-connected user.

Security concerns: Unrestricted database access

The integrated login feature works using the login control system of Windows in place of the SQL Anywhere security system to connect to a database without providing a user ID or password. Essentially, the user passes through the database security if they can log in to the computer hosting the database.

If the user successfully logs in to the Windows server as dsmith, they can connect to the database without further proof of identification provided there is either an integrated login mapping or a default integrated login user ID.

When using integrated logins, database administrators should give special consideration to the way Windows enforces login security to prevent unwanted access to the database.

Caution
Leaving the user profile Guest enabled can permit unrestricted access to a database that is hosted by that server.

If the Guest user profile is enabled and has a blank password, any attempt to log in to the server will be successful. It is not required that a user profile exist on the server, or that the login ID provided has domain login permissions. Literally any user can log in to the server using any login ID and any password: they are logged in by default to the Guest user profile.

This has important implications for connecting to a database with the integrated login feature enabled.

Consider the following scenario, which assumes the Windows server hosting a database has a Guest user profile that is enabled with a blank password.

- An integrated login mapping exists between the user fran_whitney and the database user ID DBA. When the user fran_whitney connects to the server with her correct login ID and password, she connects to the database as DBA, a user with full administrative rights.

  But anyone else attempting to connect to the server as fran_whitney will successfully log in to the server regardless of the password they provide because Windows will default that connection attempt

to the Guest user profile. Having successfully logged in to the server using the fran_whitney login ID, the unauthorized user successfully connects to the database as DBA using the integrated login mapping.

Disable the Guest user profile for security
The safest integrated login policy is to disable the Guest user profile on any Windows computer hosting a SQL Anywhere database. This can be done using the Windows User Manager utility.

Kerberos authentication

The Kerberos login feature allows you to maintain a single user ID and password for database connections, operating system, and network logins. The Kerberos login is more convenient for users and permits a single security system for database and network security. Its advantages include:

- The user does not need to provide a user ID or password to connect to the database.
- Multiple users can be mapped to a single database user ID.
- The name and password used to log in to Kerberos do not have to match the database user ID and password.

Kerberos is a network authentication protocol that provides strong authentication and encryption using secret-key cryptography. Users already logged in to Kerberos can connect to a database without providing a user ID or password.

Kerberos can be used for authentication. To delegate authentication to Kerberos you must:

- configure the server and database to use Kerberos logins
- create mapping between the user ID that logs in to the computer or network, and the database user

Caution
There are important security implications to consider when using Kerberos logins as a single security solution. See "Security concerns: Copied database files" on page 126.

SQL Anywhere does not include the Kerberos software; it must be obtained separately. The following components are included with the Kerberos software:

- **Kerberos libraries** These are referred to as the Kerberos Client or GSS (Generic Security Services)-API runtime library. These Kerberos libraries implement the well-defined GSS-API. The libraries are required on each client and server computer that intends to use Kerberos. The built-in Windows SSPI interface can be used instead of a third-party Kerberos client library if you are using Active Directory as your KDC.

  SSPI can only be used by SQL Anywhere clients in the Kerberos connection parameter. SQL Anywhere database servers cannot use SSPI—they need a supported Kerberos client other than SSPI.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **A Kerberos Key Distribution Center (KDC) server** The KDC functions as a storehouse for users and servers. It also verifies the identification of users and servers. The KDC is typically installed on a server computer not intended for applications or user logins.

SQL Anywhere supports Kerberos authentication from DBLib, ODBC, OLE DB, and ADO.NET clients, and Sybase Open Client and jConnect clients. Kerberos authentication can be used with SQL Anywhere transport layer security encryption, but SQL Anywhere does not support Kerberos encryption for network communications.

Windows uses Kerberos for Windows domains and domain accounts. Active Directory Windows Domain Controllers implement a Kerberos KDC. A third-party Kerberos client or runtime is still required on the database server computer for authentication in this environment, but the Windows client computers can use the built-in Windows SSPI interface instead of a third-party Kerberos client or runtime. See "Use SSPI for Kerberos logins on Windows" on page 122.

Kerberos clients

Kerberos authentication is available on 32-bit Windows and Linux. For a list of tested Kerberos clients, see http://www.sybase.com/detail?id=1061807.

The following table lists the default names and locations of the keytab and GSS-API files used by the supported Kerberos clients.

Note
SSPI can only be used by SQL Anywhere clients in the Kerberos connection parameter. SQL Anywhere database servers cannot use SSPI—they need a supported Kerberos client other than SSPI.

Kerberos client	Default keytab file	GSS-API library file name	Notes
Windows MIT Kerberos client	*C:\WINDOWS\krb5kt*	*gssapi32.dll*	The KRB5_KTNAME environment variable can be set before starting the database server to specify a different keytab file.
Windows CyberSafe Kerberos client	*C:\Program Files\CyberSafe\v5srvtab*	*gssapi32.dll*	The CSFC5KTNAME environment variable can be set before starting the database server to specify a different keytab file.

Kerberos client	Default keytab file	GSS-API library file name	Notes
Unix MIT Kerberos client	*/etc/krb5.keytab*	*libgssapi_krb5.so* [1]	The KRB5_KTNAME environment variable can be set before starting the database server to specify a different keytab file.
Unix CyberSafe Kerberos client	*/krb5/v5srvtab*	*libgss.so* [1]	The CSFC5KTNAME environment variable can be set before starting the database server to specify a different keytab file.
Unix Heimdal Kerberos client	*/etc/krb5.keytab*	*libgssapi.so.1* [1]	

[1] These file names may vary depending on your operating system and Kerberos client version.

Set up Kerberos authentication

To set up Kerberos authentication on a SQL Anywhere database

1. Install and configure the Kerberos client software, including the GSS-API runtime library, on both the client and server.

   On Windows client computers using an Active Directory KDC, SSPI can be used and you do not need to install the Kerberos client. See "Use SSPI for Kerberos logins on Windows" on page 122.

2. If necessary, create a Kerberos principal in the Kerberos Key Distribution Center (KDC) for each user.

   A Kerberos principal is a Kerberos user ID in the format *user*/*instance*@*REALM*, where /*instance* is optional. If you are already using Kerberos, the principal should already exist, so you will not need to create a Kerberos principal for each user.

   Principals are case sensitive and must be specified in the correct case. Mappings for multiple principals that differ only in case are not supported (for example, you cannot have mappings for both jjordan@MYREALM.COM and JJordan@MYREALM.COM).

3. Create a Kerberos principal in the KDC for the SQL Anywhere database server.

   The Kerberos principal for the database server has the format *server-name*@*REALM*, where *server-name* is the SQL Anywhere database server name. Principals are case significant, and the *server-name* cannot contain multibyte characters, or the characters /, \, or @. The rest of the steps assume the Kerberos principal is my_server_princ@MYREALM.COM.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You must create a server service principal within the KDC because servers use a keytab file for KDC authentication. The keytab file is protected and encrypted.

4. Securely extract and copy the keytab for the principal *server-name@REALM* from the KDC to the computer running the SQL Anywhere database server. The default location of the keytab file depends on the Kerberos client and the platform. The keytab file's permissions should be set so that the SQL Anywhere server can read it, but unauthorized users do not have read permission.

5. Configure SQL Anywhere to use Kerberos

Configure SQL Anywhere to use Kerberos

To configure a SQL Anywhere database to use Kerberos

1. Set up Kerberos authentication on the SQL Anywhere database. See "Set up Kerberos authentication" on page 118.

2. Start the SQL Anywhere server with the -krb or -kr option to enable Kerberos authentication, or use the -kl option to specify the location of the GSS-API library and enable Kerberos.

3. Change the public or temporary public option login_mode to a value that includes Kerberos. You must have DBA authority to change the setting of this option. The login_mode database option determines whether Kerberos logins are allowed. As database options apply only to the database in which they are found, different databases can have a different Kerberos login setting, even if they are loaded and running on the same server. For example:

   ```
   SET OPTION PUBLIC.login_mode = 'Kerberos,Standard';
   ```

   The login_mode database option accepts one or more of the following values:

   - **Standard** Standard logins are permitted. This value is the default. Standard connection logins must supply both a user ID and password, and do not use the Integrated or Kerberos connection parameters.
   - **Integrated** Integrated logins are permitted.
   - **Kerberos** Kerberos logins are permitted.

   Caution
   Setting the login_mode database option to Kerberos restricts connections to only those users who have been granted a Kerberos login mapping. Attempting to connect using a user ID and password generates an error unless you are a user with DBA authority.

4. Create a database user ID for the client. You can use an existing database user ID for the Kerberos login, as long as that user has the correct permissions. For example:

   ```
   CREATE USER "kerberos-user"
   IDENTIFIED BY abc123;
   ```

5. Execute a GRANT KERBEROS LOGIN TO statement to create a mapping from the client's Kerberos principal to an existing database user ID. This statement requires DBA authority. For example:

```
GRANT KERBEROS LOGIN TO "pchin@MYREALM.COM"
AS USER "kerberos-user";
```

If you want to connect when a Kerberos principal is used that does not have a mapping, ensure the Guest database user ID exists and has a password. See “Creating a default integrated login user” on page 114.

6. Ensure the client user has already logged on (has a valid Kerberos ticket-granting ticket) using their Kerberos principal and that the client's Kerberos ticket has not expired. A Windows user logged in to a domain account already has a ticket-granting ticket, which allows them to authenticate to servers, providing their principal has enough permissions.

   A ticket-granting ticket is a Kerberos ticket encrypted with the user's password that is used by the Ticket Granting Service to verify the user's identity.

7. Connect from the client, specifying the KERBEROS connection parameter (Often KERBEROS=YES, but KERBEROS=SSPI or KERBEROS=*GSS-API-library-file* can also be used). If the user ID or password connection parameters are specified, they are ignored. For example:

   ```
   dbisql -c "KERBEROS=YES;Server=my_server_princ"
   ```

Interactive SQL example

For example, a connection attempt using the following Interactive SQL statement is successful if the user logs in with a user profile name that matches a Kerberos login mapping in a default database of a server:

```
CONNECT USING 'KERBEROS=YES';
```

The Interactive SQL statement CONNECT can connect to a database if all the following are true:

- A server is currently running.
- The default database on the current server is enabled to accept Kerberos authenticated connections.
- A Kerberos login mapping has been created for the user's current Kerberos principal.
- If the user is prompted with a window by the database server for more connection information (such as occurs when using Interactive SQL), the user clicks OK without providing more information.

See also

- “-kl dbeng12/dbsrv12 server option” on page 201
- “-kr dbeng12/dbsrv12 server option” on page 202
- “-krb dbeng12/dbsrv12 server option” on page 203
- “login_mode option” on page 547
- “GRANT statement” [*SQL Anywhere Server - SQL Reference*]
- “Create Kerberos login mappings” on page 121

Connect from a Sybase Open Client or jConnect application

To connect from a Sybase Open Client or jConnect application:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- Set up Kerberos authentication. See "Set up Kerberos authentication" on page 118.
- Configure SQL Anywhere to use Kerberos. See "Configure SQL Anywhere to use Kerberos" on page 119.
- Set up Sybase Open Client or jConnect as you would for Kerberos authentication with Adaptive Server Enterprise. The server name must be the SQL Anywhere server's name and is case significant. You cannot connect using an alternate server name from Sybase Open Client or jConnect.

For information about setting up the Kerberos principals and extracting the keytab, see http://www.sybase.com/detail?id=1029260.

See also

- "-krb dbeng12/dbsrv12 server option" on page 203
- "-kr dbeng12/dbsrv12 server option" on page 202
- "-kl dbeng12/dbsrv12 server option" on page 201
- "login_mode option" on page 547
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE USER statement" [*SQL Anywhere Server - SQL Reference*]
- "Kerberos (KRB) connection parameter" on page 294
- "Troubleshooting Kerberos connections" on page 123

Create Kerberos login mappings

To create a Kerberos login mapping (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, right-click **Login Mappings** and choose **New** » **Login Mapping**.
3. Follow the instructions in the **Create Login Mapping Wizard**.

To create a Kerberos login mapping (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a GRANT KERBEROS LOGIN TO statement.

   See "GRANT statement" [*SQL Anywhere Server - SQL Reference*].

Example

The following SQL statement grants KERBEROS login permission to the Windows user pchin.

```
GRANT KERBEROS LOGIN TO "pchin@MYREALM.COM"
AS USER "kerberos-user";
```

Revoke Kerberos login permission

To revoke a Kerberos login mapping (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, click **Login Mappings**.
3. In the right pane, right-click the login mapping and choose **Delete**.
4. Click **Yes**.

To revoke a Kerberos login mapping (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a REVOKE KERBEROS LOGIN FROM statement.

   See "REVOKE statement" [*SQL Anywhere Server - SQL Reference*].

Example

The following SQL statement removes KERBEROS login permission from the Windows user pchin.

```
REVOKE KERBEROS LOGIN
FROM "pchin@MYREALM.COM";
```

Use SSPI for Kerberos logins on Windows

In a Windows domain, SSPI can be used on Windows-based computers without a Kerberos client installed on the client computer. Windows domain accounts already have associated Kerberos principals.

Note
SSPI can only be used by SQL Anywhere clients in the Kerberos connection parameter. SQL Anywhere database servers cannot use SSPI—they need a supported Kerberos client other than SSPI.

To connect using SSPI

1. Set up Kerberos authentication. See "Set up Kerberos authentication" on page 118.
2. Start the SQL Anywhere server with the -krb option to enable Kerberos authentication. For example:

   ```
   dbeng12 -krb -n my_server_princ C:\kerberos.db
   ```

3. Change the public or temporary public option login_mode to a value that includes Kerberos. You must have DBA authority to set this option. For example:

   ```
   SET OPTION PUBLIC.login_mode = 'Kerberos';
   ```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

4. Create a database user ID for the client. You can use an existing database user ID for the Kerberos login, as long as that user has the correct permissions. For example:

   ```
   CREATE USER kerberos_user
   IDENTIFIED BY abc123;
   ```

5. Create a mapping from the client's Kerberos principal to an existing database user ID by executing a GRANT KERBEROS LOGIN TO statement. This statement requires DBA authority. For example:

   ```
   GRANT KERBEROS LOGIN TO "pchin@MYREALM.COM"
   AS USER "kerberos-user";
   ```

6. Connect to the database from the client computer. For example:

   ```
   dbisql -c "KERBEROS=SSPI;Server=my_server_princ"
   ```

   When Kerberos=SSPI is specified in the connection string, a Kerberos login is attempted.

   A connection attempt using the following Interactive SQL statement will also succeed, providing the user has logged on with a user profile name that matches a Kerberos login mapping in a default database of a server:

   ```
   CONNECT USING 'KERBEROS=SSPI';
   ```

Troubleshooting Kerberos connections

If you get unexpected errors when attempting to enable or use Kerberos authentication, it is recommended that you enable additional diagnostic messages on the database server and client.

Specifying the -z option when you start the database server, or using `CALL sa_server_option( 'DebuggingInformation', 'ON' )` if the server is already running includes additional diagnostic messages in the database server message log. The LogFile connection parameter writes client diagnostic messages to the specified file. As an alternative to using the LogFile connection parameter, you can execute the command dbping -z. The -z parameter displays diagnostic messages that should help identify the cause of the connection problem. See “-z dbeng12/dbsrv12 server option” on page 243.

Difficulties starting the database server

Symptom	Common solutions
"Unable to load Kerberos GSS-API library" message	• Ensure a Kerberos client is installed on the database server computer, including the GSS-API library. • The database server -z output lists the name of the library that it is attempting to load. Verify the library name is correct. If necessary, use the -kl option to specify the correct library name. • Ensure the directory and any supporting libraries is listed in the library path (%PATH% on Windows). • If the database server -z output states the GSS-API library was missing entry points, then the library is not a supported Kerberos Version 5 GSS-API library.
"Unable to acquire Kerberos credentials for server name "*server-name*"" message	• Ensure there is a principal for *server-name@REALM* in the KDC. Principals are case sensitive, so ensure the database server name is in the same case as the user portion of the principal name. • Ensure the name of the SQL Anywhere server is the primary/user portion of the principal. • Ensure that the server's principal has been extracted to a keytab file and the keytab file is in the correct location for the Kerberos client. See "Kerberos clients" on page 117. • If the default realm for the Kerberos client on the database server computer is different from the realm in the server principal, use the -kr option to specify the realm in the server principal.
"Kerberos login failed" client error	• Check the database server diagnostic messages. Some problems with the keytab file used by the server are not detected until a client attempts to authenticate.

Troubleshooting Kerberos client connections

If the client got an error attempting to connect using Kerberos authentication:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Symptom	Common solutions
"Kerberos logins are not supported" error and the LogFile includes the message "Failed to load the Kerberos GSS-API library"	• Ensure a Kerberos client is installed on the client computer, including the GSS-API library. • The file specified by LogFile lists the name of the library that it is attempting to load. Verify that the library name is correct, and use the Kerberos connection parameter to specify the correct library name, if necessary. • Ensure that the directory including any supporting libraries is listed in the library path (%PATH% on Windows). • If the LogFile output states the GSS-API library was missing entry points, then the library is not a supported Kerberos Version 5 GSS-API library.
"Kerberos logins are not supported" error	• Ensure the database server has enabled Kerberos logins by specifying one or more of the -krb, -kl, or -kr server options. • Ensure Kerberos logins are supported by SQL Anywhere on both the client and server platforms.
"Kerberos login failed" error	• Ensure the user is logged into Kerberos and has a valid ticket-granting ticket that has not expired. • Ensure the client computer and server computer both have their time synchronized to within less than 5 minutes.
"Login mode 'Kerberos' not permitted by login_mode setting" error	• The public or temporary public database option setting for the login_mode option must include the value Kerberos to allow Kerberos logins.
"The login ID '*client-Kerberos-principal*' has not been mapped to any database user ID"	• The Kerberos principal must be mapped to a database user ID using the GRANT KERBEROS LOGIN statement. Note the full client principal including the realm must be provided to the GRANT KERBEROS LOGIN statement, and principals which differ only in the instance or realm are treated as different. • Alternatively, if you want any valid Kerberos principal which has not be explicitly mapped to be able to connect, create the guest database user ID with a password using GRANT CONNECT.

Security concerns: Setting temporary public options for added security

Setting the value of the login_mode option for a given database to allow a combination of standard, integrated, and Kerberos logins using the SET OPTION statement permanently enables the specified types of logins for that database. For example, the following statement permanently enables standard and integrated logins:

```
SET OPTION PUBLIC.login_mode = 'Standard,Integrated';
```

If the database is shut down and restarted, the option value remains the same and integrated logins remain enabled.

Setting the login_mode option using SET TEMPORARY OPTION still allows user access via integrated logins, but only until the database is shut down. The following statement changes the option value temporarily:

```
SET TEMPORARY OPTION PUBLIC.login_mode = 'Standard,Integrated';
```

If the permanent option value is Standard, the database will revert to that value when it is shut down.

Setting temporary public options can provide additional security for your database. When you add integrated or Kerberos logins to your database, the database relies on the security of the operating system on which it is running. If the database is copied to another computer, access to the database reverts to the SQL Anywhere security model.

See also

- “Security concerns: Copied database files” on page 126
- “SET OPTION statement” [*SQL Anywhere Server - SQL Reference*]

Security concerns: Copied database files

If the database file can be copied, use the temporary public login_mode option for integrated and Kerberos logins. If the file is copied, the integrated and Kerberos logins are not supported by default.

If a database contains sensitive information, the computer where the database files are stored should be protected from unauthorized access. Otherwise, the database files could be copied and unauthorized access to the data could be obtained on another computer. To increase database security:

- Make user passwords, especially those with DBA authority, complex and difficult to guess.
- Set the PUBLIC.login_mode database option to Standard. To enable integrated or Kerberos logins, only the temporary public option should be changed each time the server is started. This ensures that only Standard logins are allowed if the database is copied. See “Security concerns: Setting temporary public options for added security” on page 126.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- Strongly encrypt the database file using the AES encryption algorithm. The encryption key should be complex and difficult to guess.

Sample SQL Anywhere database connections

The following examples show you how to connect to a SQL Anywhere database from the tools included with SQL Anywhere.

Connect to the sample database from Sybase Central or Interactive SQL

To connect to the sample database (Sybase Central)

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. Click **Connections** » **Connect With SQL Anywhere 12**.
3. From the **Action** dropdown list choose **Connect With An ODBC Data Source**.
4. Click **ODBC Data Source Name**, and then type in the box below, **SQL Anywhere 12 Demo**.
5. Click **Connect**.

To connect to the sample database (Interactive SQL)

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Interactive SQL**.
2. Click **Change Database Type** and choose **SQL Anywhere**.
3. From the **Action** dropdown list choose **Connect With An ODBC Data Source**
4. Click **ODBC Data Source Name**, and then type in the box below, SQL Anywhere 12 Demo.
5. Click **Connect**.

Note
You do not need to enter a user ID and a password for this connection because the data source already contains this information.

However, in a production environment, it is recommended that you do not store passwords in the ODBC Data Source. See "Do not include passwords in ODBC data sources" on page 1119.

To connect to the sample database (specifying the database file location)

1. In Sybase Central or Interactive SQL, open the **Connect** window. See "Open the Connect window" on page 95.
2. From the **Authentication** dropdown list, choose **Database**.
3. In the **User ID** field, type **DBA**.
4. In the **Password** field, type **sql**.
5. From the **Action** dropdown list, choose **Connect To A Database On This Computer** to connect to a database that is running on your computer.
6. In the **Database File** field, browse to *samples-dir*. On Microsoft Windows XP operating systems the default location is *C:\Documents and Settings\All Users\Shared Documents\SQL Anywhere 12\Samples\demo.db*.

   For information about *samples-dir*, see "Samples directory" on page 392.
7. In the **Database Name** field, type **demo.db**.
8. Click **Connect**.

See also

- "Tutorial: Using the sample database" on page 1
- "Working with the Connect window" on page 92

Connect to the sample database on Unix

The following procedure describes how to connect to the sample database from Interactive SQL, and how to execute a query.

To connect to the sample database from Interactive SQL (Unix)

1. In a typical Unix installation, the SQL Anywhere software, including the sample database, is installed into a directory that you do not have permissions to write to. Before you begin, in a terminal window, change to a writable folder. Copy the sample database file into the folder by using the following command:

   ```
   cp install-dir/demo.db  .
   ```

   install-dir is the directory where SQL Anywhere 12 is installed, for example, */opt/sqlanywhere12*.
2. Start the sample database on a database server:

   ```
   dbeng12 demo.db
   ```
3. In another terminal window, start Interactive SQL:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
dbisql
```

If you have not installed the JRE or if you are using the Deployment option, you can run the character-based version of Interactive SQL by running the following command:

```
dbisqlc
```

A connection window appears.

4. Type **DBA** as the **User ID**, **sql** as the **Password**, and **demo** as the **Database Name**. Leave the other fields blank. Press Enter to connect to the database.

See also

- "Tutorial: Using the sample database" on page 1
- "Working with the Connect window" on page 92

Connect to the sample database on Mac OS X

To connect to the sample database from Interactive SQL (Mac OS X)

1. In the Finder, locate the SQL Anywhere sample database. By default, it is located in */Applications/SQLAnywhere12/demo.db*.
2. Copy this file to a location where you have read and write access, such as the Desktop.
3. In the Finder, double-click **DBLauncher**.

   By default, **DBLauncher** is located at the following path: */Applications/SQLAnywhere12*.
4. Start a new server.
   a. In the **Database** field, browse to the location of the SQL Anywhere sample database. (For example, */Users/user-id/Desktop/demo.db*.
   b. In the **Server Name** field, type **demo**.
   c. Select **Local Server**.

      The **Local Server** option does not allow client/server communications over a network.
   d. Click **Start** to start a personal database server named **demo**.
5. In the Finder, double-click **Interactive SQL** in */Applications/SQLAnywhere12*.
6. Connect to the SQL Anywhere sample database.
   - From the **Authentication** dropdown list, choose **Database**.
   - In the **User ID** field, type **DBA**.
   - In the **Password** field, type **sql**.

- From the **Action** dropdown list choose **Connect To A Database On This Computer** to connect to the database that is running on your computer.
- In the **Server Name** field, type **demo**.
- In the **Database Name** field, type **demo**.
- Click **Connect**.

See also

- "Tutorial: Using the sample database" on page 1
- "Working with the Connect window" on page 92

Connect to a local database

Use one of the following procedures to connect to a database residing on your computer. If the database is already loaded (started) on the server, only the database name is required to connect to the database. You do not need to specify a database file.

To simplify database access, use a connection profile. See "Sybase Central connection profiles" on page 95.

To connect to a database that is already running on a local server (Sybase Central and Interactive SQL)

1. Start Sybase Central or Interactive SQL.

   If the **Connect** window does not appear:

   - In Sybase Central, choose **Connections** » **Connect With SQL Anywhere 12**.
   - In Interactive SQL, choose **SQL** » **Connect**.

     Click **Change Database Type**, and then choose **SQL Anywhere**.

2. From the **Authentication** dropdown list, choose **Database**.
3. In the **User ID** field, type a user name.
4. In the **Password** field, type a password for the database.
5. From the **Action** dropdown list, choose **Connect To A Database On This Computer** to connect to a database that is running on your computer.
6. If the server is running a single database, click **Connect**.

   If the server is running multiple databases:

   - In the **Database Name** field, type the name of the database.
   - Click **Connect**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To start and connect to a database (Sybase Central and Interactive SQL)

1. Start Sybase Central or Interactive SQL.

   If the **Connect** window does not appear:

   - In Sybase Central, choose **Connections** » **Connect With SQL Anywhere 12**.
   - In Interactive SQL, choose **SQL** » **Connect**.
   - Click **Change Database Type**, and then choose **SQL Anywhere**.

2. From the **Authentication** dropdown list, choose **Database**.
3. In the **User ID** field, type a user name.
4. In the **Password** field, type a password for the database.
5. From the **Action** dropdown list, choose **Start And Connect To A Database On This Computer** to start and connect to a database on your computer.
6. In the **Database File** field, specify the file path, file name, and file extension.
7. To create a database name that is different from the file name for subsequent connections, type a name in the **Database Name** field. Do not specify a file path or extension.
8. Click **Connect**.

Connecting to an embedded database

An **embedded database**, designed for use by a single application, runs on the same computer as the application and is generally hidden from the user.

When an application uses an embedded database, the personal server is generally not running when the application connects. The database is started using the connection string, and by specifying the database file in the DatabaseFile (DBF) parameter of the connection string.

To improve query performance for databases that are started automatically, start the database as soon as possible, even if users are not connecting right away. This allows the cache to warm before queries are executed against the database. See "Using cache warming" [*SQL Anywhere Server - SQL Usage*].

Using the DBF connection parameter

The DBF connection parameter specifies the database file to use. The database file automatically loads onto the default server, or starts a server if none are running.

The database unloads when there are no more connections to the database (generally when the application that started the connection disconnects). If the connection started the server, the database server stops once the database unloads.

In the following example, the sample database is loaded as an embedded database:

```
DBF=samples-dir\demo.db
```

For information about *samples-dir*, see "Samples directory" on page 392.

Using the ServerName (Server) connection parameter

When using an embedded database it is recommended that you use the ServerName (Server) connection parameter. This ensures that the database connects to the correct database server if there are other applications running SQL Anywhere database servers on the same computer.

Using the StartLine [START] connection parameter

The following connection parameters show you how to customize the startup of the sample database as an embedded database. This is useful if you want to use options, such as the cache size:

```
START=dbeng12 -c 8M
DBF=samples-dir\demo.db
```

There are many connection parameters that affect how a server is started. It is recommended that you use the following connection parameters instead of providing the corresponding server options within the StartLine (START) connection parameter:

- ServerName (Server)
- DatabaseFile (DBF)
- DatabaseSwitches (DBS)
- DatabaseName (DBN)

Using the ELEVATE connection parameter

If you are automatically starting a database server on Windows Vista, you must specify ELEVATE=YES in your connection string so that the database server executables are elevated. On Windows Vista, only elevated database servers can use AWE memory or call procedures as an administrator user.

Note
The use of AWE is deprecated. It is recommended that you use the 64-bit version of the SQL Anywhere database server on a 64-bit Windows operating system if you require a large cache.

See also

- "DatabaseFile (DBF) connection parameter" on page 280
- "ServerName (Server) connection parameter" on page 306
- "StartLine (START) connection parameter" on page 308
- "Elevate connection parameter" on page 286
- "Open the Connect window" on page 95
- "Sample SQL Anywhere database connections" on page 127

Connect using a data source

You can save sets of connection parameters in a **data source**. All SQL Anywhere interfaces, except Sybase Open Client and jConnect, can use data sources.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To connect using a data source (Sybase Central or Interactive SQL)

1. Start Sybase Central or Interactive SQL.

   If the **Connect** window does not appear:

   - In Sybase Central, choose **Connections** » **Connect With SQL Anywhere 12**.
   - In Interactive SQL, choose **SQL** » **Connect**.

     Click **Change Database Type**, and then choose **SQL Anywhere**.

2. From the **Authentication** dropdown list, choose **Database**.
3. In the **User ID** field, type a user name.
4. In the **Password** field, type a password for the database.
5. From the **Action** dropdown list, choose **Connect Using An ODBC Data Source**.
6. Do one of the following:
   - Click **ODBC Data Source Name** and enter the DataSourceName (DSN) connection parameter that references a data source in the Windows registry. Click **Browse** to view a list of data sources.
   - Click **ODBC Data Source File** and enter the FileDataSourceName (FILEDSN) connection parameter that references a data source held in a file. Click **Browse** to view a list of files.
7. Click **Connect**.

To connect using a data source with a connection string (Sybase Central or Interactive SQL

1. Start Sybase Central or Interactive SQL.

   If the **Connect** window does not appear:

   - In Sybase Central, choose **Connections** » **Connect With SQL Anywhere 12**.
   - In Interactive SQL, choose **SQL** » **Connect**.

     Click **Change Database Type**, and then choose **SQL Anywhere**.

2. From the **Authentication** dropdown list, choose **Database**.
3. In the **User ID** field, type a user name.
4. In the **Password** field, type a password for the database.
5. From the **Action** dropdown list, choose **Connect With A Connection String**.
6. In the **Parameters** field, type connection parameters in a semicolon delimited list of parameter=value pairs. For example:

   ```
   DSN=SQL Anywhere 12 Demo";Server=SampleServer
   ```

7. Click **Connect**.

See also

- "Open the Connect window" on page 95
- "Sample SQL Anywhere database connections" on page 127
- "Using ODBC data sources on Unix" on page 105

Connect to a database server on a network

To connect to a database on a network server (Sybase Central or Interactive SQL)

1. Start Sybase Central or Interactive SQL.

   If the **Connect** window does not appear:

   - In Sybase Central, choose **Connections** » **Connect With SQL Anywhere 12**.
   - In Interactive SQL, choose **SQL** » **Connect**.

     Click **Change Database Type**, and then choose **SQL Anywhere**.

2. From the **Authentication** dropdown list, choose **Database**.
3. In the **User ID** field, type a user name.
4. In the **Password** field, type a password for the database.
5. From the **Action** dropdown list, choose **Connect To A Running Database On Another Computer** to connect to a database that is running on another computer.
6. In the **Server Name** field, type the name of the server or click **Find**.
7. In the **Database Name** field, type the name of the database.
8. Click **Connect**.

Using default connection parameters

You can use default behavior to make a connection and leave the connection parameters unspecified. However, using the default behavior in a production environment can cause problems if the application is installed with other SQL Anywhere applications. For more information about default behavior, see "Troubleshooting connections" on page 138.

Default database server and database

Use the default parameters to connect to a single personal server with a single database:

```
UID=user-id
PWD=password
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Default database server

If more than one database is on a single personal server, use the default server settings, and specify the database you want to connect to:

```
DBN=db-name
UID=user-id
PWD=password
```

Default database

If more than one server is running, specify which server you want to connect to. You do not need to specify the database name if only a single database is on that server. The following connection string connects to a named server, using the default database:

```
Server=server-name
UID=user-id
PWD=password
```

Default database server on another computer

The following connection string connects to a database running on a different computer:

```
Host=host-name
DBN=db-name
UID=user-id
PWD=password
```

No defaults for a local server

The following connection string connects to a named local server, using a named database:

```
Server=server-name
DBN=db-name
UID=user-id
PWD=password
```

No defaults for a network server

To connect to a network server running on a different computer:

```
Host=host-name
Server=server-name
DBN=dbn
UID=user-id
PWD=password
```

If Host is not specified, only local shared memory connections are attempted.

If you are connecting from Sybase Central, Interactive SQL, or the SQL Anywhere Console utility (dbconsole), you can choose the **Connect To A Running Database On Another Computer** option on the **Connect** window to attempt a network connection.

Connecting from SQL Anywhere utilities

All SQL Anywhere database utilities use embedded SQL to communicate with the server.

How database utilities obtain connection parameter values

Many of the administration utilities obtain the connection parameter values by:

1. Using values specified on the command line. For example, the following command starts a backup of the default database on the default server using the user ID DBA and the password sql:

   ```
   dbbackup -c "UID=DBA;PWD=sql" c:\backup
   ```

   For more information about the options for each database utility, see "Database administration utilities" on page 763.

2. Using the SQLCONNECT environment variable settings if any values are missing. SQL Anywhere does not set this variable automatically.

   See "SQLCONNECT environment variable" on page 389.

SQL Anywhere connection pooling

Connection pooling may improve the performance of applications that make multiple, brief connections to the database server. If connection pooling is enabled for a connection, when it is disconnected, it is automatically cached and may be reused when the application reconnects. You control connection pooling with the ConnectionPool (CPOOL) connection parameter. Once an application makes a specified number of connections with the same connection string, then the connection is pooled.

An application must make five connections with the same connection string before a connection is cached. The connection name can be unique each time, but all other connection parameters must be identical for a cached connection to be reused.

If the application process connects again and there are cached connections available for the same connection string, the cached connection is reused. Connections remain in the cached state for the time specified by the ConnectionPool (CPOOL) connection parameter (60 seconds by default).

Cached connections are not reused if it would change the behavior of the application. For example, cached connections are not reused for databases that stop automatically when there are no connections to them, if connections are disabled, if the database server has reached its connection limit, or if a password has changed.

To ensure that connection pooling is transparent to the application, a connection is disconnected if a failure occurs when caching a connection. If a failure occurs when attempting to reuse a cached connection, the database server attempts to connect normally.

A connection is cached if it is disconnected and the maximum number of connections specified by the CPOOL connection parameter has not been reached. The connection is reinitialized, and the cached connection remains connected to the database server even though the application has disconnected it. The cleanup and reinitialization of a connection includes the following activities:

- Rolling back all outstanding transactions.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- Dropping temporary tables, temporary functions, and variables.
- Resetting connection options and connection counters.
- Decrementing and incrementing the database server connection counts. You are not informed that there are active connections when a database server with cached connections shuts down.
- Executing all defined disconnect and connect events.
- Executing the login_procedure database option and verifying the login policy.
- Resetting the connection ID.

Using SQL Anywhere connection pooling with other connection pooling products

If you are using a product or API that supports connection pooling, then the connection pooling of the product or API supersedes SQL Anywhere connection pooling. Both types of connection pooling can be active at the same time.

The behavior of connection pooling in your product or the API may be significantly different than SQL Anywhere connection pooling. If the behavior of connection pooling in your product or API is inappropriate for an application, SQL Anywhere connection pooling can be used and may improve the performance of some applications.

Connection pooling and read-only scale-out

If the NodeType (NODE) connection parameter is also specified for a connection, the application typically connects to the primary server and the primary server determines which copy server is least heavily loaded. The connection is then redirected to that node. If the application makes and drops several such connections within a short period of time, the connection is pooled and the primary server is not asked which copy server to use. This behavior reduces the load on the primary server, but may not give expected behavior.

See also

- “ConnectionPool (CPOOL) connection parameter” on page 279
- “Using connection parameters” on page 86
- “ConnPoolCachedCount database property” on page 663
- “ConnPoolHits database property” on page 663
- “ConnPoolMisses database property” on page 663

Temporary connections

The SQL Anywhere database server uses temporary connections to perform operations such as running backups or initializing databases. You can get information about temporary connections by using the sa_conn_info or sa_conn_list system procedure. The ParentConnection property returns the connection ID of the connection that spawned the temporary connection. See “ParentConnection connection property” on page 634.

Temporary connections have connection IDs that are larger than 1 billion (1000000000), and their names describe the function of the connection. For a list of temporary connection names, see "Name connection property" on page 632.

The following example uses the sa_conn_info system procedure to return a result set showing which connection created a temporary connection.

```
SELECT Number, Name, ParentConnection FROM sa_conn_info();
```

Connection 8 spawned the temporary connection that executed a CREATE DATABASE statement.

```
Number        Name               ParentConnection
------------------------------------------------------------
1000000048    INT: CreateDB      8
9             SQL_DBC_14675af8   (NULL)
8             SQL_DBA_152d5ac0   (NULL)
```

See also

- "sa_conn_info system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_conn_list system procedure" [*SQL Anywhere Server - SQL Reference*]
- "CONNECTION_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]

Troubleshooting connections

An understanding of how SQL Anywhere establishes connections can help you resolve connectivity problems. For information about network-specific issues, including connections across firewalls, see "Client/server communications" on page 76.

To establish a connection, SQL Anywhere:

- Locates the interface library
- Assembles a list of connection parameters
- Locates a server
- Locates the database
- Starts a personal server when the database server is not located

The SQL Anywhere connection procedure is the same for:

- Any **ODBC application** using the SQLDriverConnect function, which is the common connection method for ODBC applications. Many application development systems, such as Sybase PowerBuilder, belong to this class of application. The SQLConnect function is also available to ODBC applications.
- Any client application using **embedded SQL** and using the recommended function for connecting to a database (db_string_connect). In addition, the SQL CONNECT statement is available for embedded

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

SQL applications and in Interactive SQL. It has two forms: CONNECT AS ... and CONNECT USING. All the database administration tools, including Interactive SQL, use db_string_connect.

- Any **ADO application** using the ADODB Connection object. The Provider property is used to locate the OLE DB driver. The Connection String property may use **DataSource** as an alternative to **DataSourceName** and **User ID** as an alternative to **UserID**.
- Any application using the **SQL Anywhere JDBC driver** to pass the URL **jdbc:sqlanywhere:** followed by a standard connection string as a parameter to the Driver Manager.GetConnection method.

See also

- "Troubleshooting database server startup" on page 998
- "Troubleshooting network communications" on page 1002

Locating the interface library

Generally, the location of this DLL or shared library is transparent to the user.

ODBC driver location

For ODBC, the interface library is also called an ODBC driver. An ODBC client application calls the ODBC driver manager, and the driver manager locates the SQL Anywhere driver.

The ODBC driver manager searches the supplied data source to locate the driver. When you create a data source using the ODBC Data Source Administrator or dbdsn utility, SQL Anywhere fills in the current location for your ODBC driver. The data source information is stored in the Windows registry, or in the Unix system information file (named *.odbc.ini* by default).

Embedded SQL interface library location

Embedded SQL applications call the interface library by name. The name of the SQL Anywhere embedded SQL interface library is:

- **Windows** *dblib12.dll*
- **Unix** *libdblib12* with an operating-system-specific extension

OLE DB driver location

The provider name (SAOLEDB) is used to locate the SQL Anywhere OLE DB Provider DLL (*dboledb12.dll)* based on entries in the registry. The entries are created when the SAOLEDB provider is installed or if it is re-registered.

ADO.NET

ADO.NET programs add a reference to the SQL Anywhere ADO.NET provider, which is named *iAnywhere.Data.SQLAnywhere.dll*. The .NET Data Provider DLL is added to the .NET Global Assembly Cache (GAC) when it is installed.

SQL Anywhere JDBC driver location

When you run your application, the Java package *sajdbc.jar* (JDBC 3.0) or *sajdbc4.jar* (JDBC 4.0) must be in the classpath. The system must be able to locate the native DLLs or shared objects.

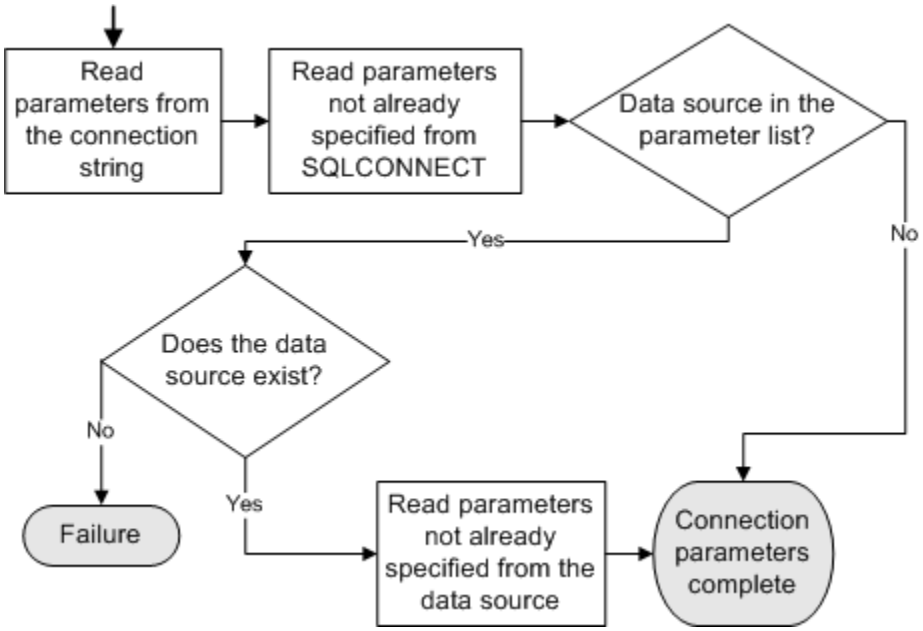
- **PC operating systems** On PC operating systems such as Windows, the current directory, the system path, and in the *Windows* and *Windows\system32* directories are searched.
- **Unix operating systems** On Unix, the system path and the user library path are searched.

When the library is located

A connection string is sent to the interface library when it is located by the client application. The string is used by the interface library to assemble a list of connection parameters, and establish a server connection.

Assembling a list of connection parameters

The following diagram illustrates how the interface library assembles the list of connection parameters and establishes a connection.



- **Precedence** Parameters held in more than one place are subject to the following order of precedence:

  1. Connection string
  2. SQLCONNECT
  3. Data source

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

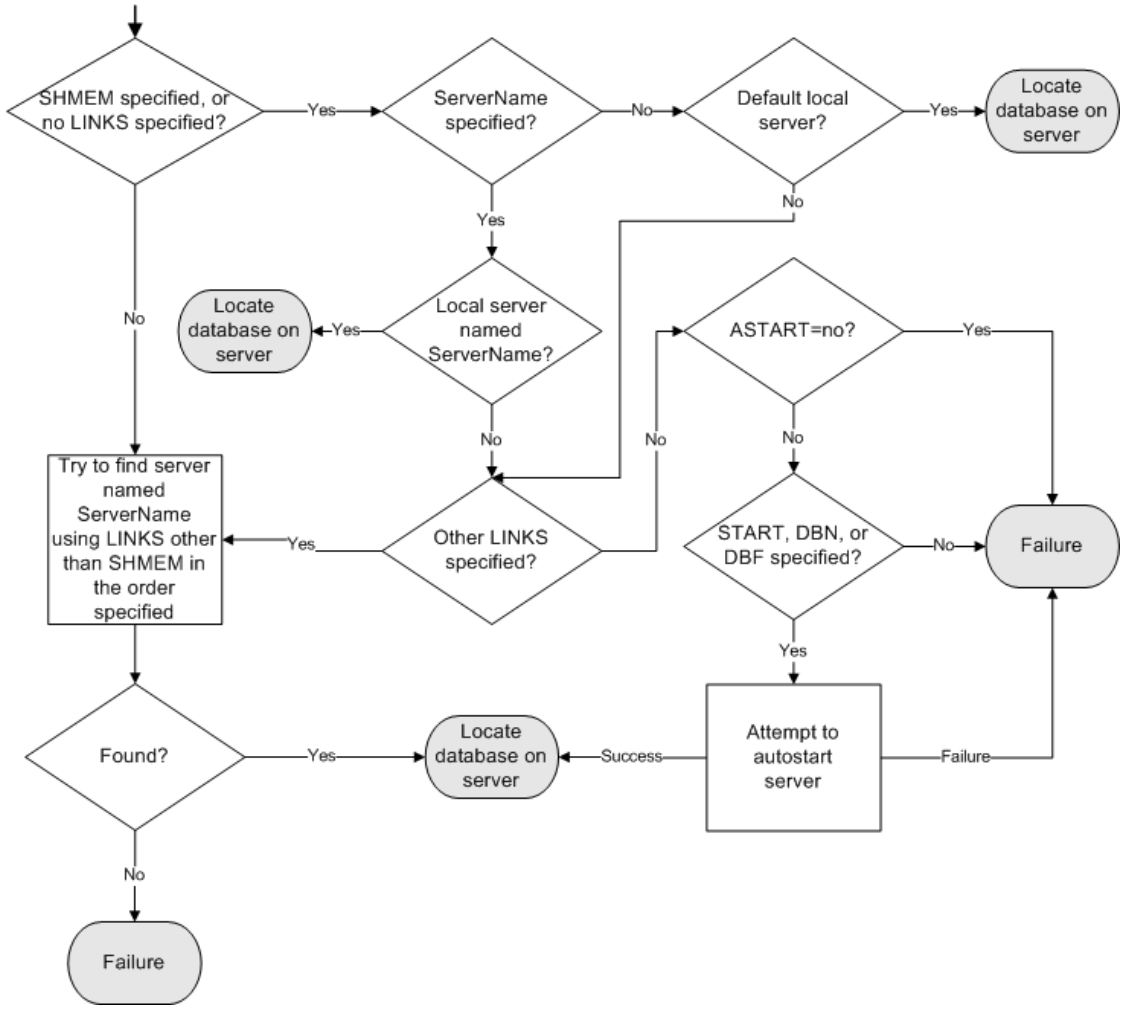
If a parameter is supplied both in a data source and in a connection string, the connection string value overrides the data source value.

- **Failure** Failure at this stage occurs only if you specify in the connection string or in SQLCONNECT a data source that does not exist.
- **Common parameters** Depending on other connections already in use, some connection parameters may be ignored, including:
  - **AutoStop** Ignored if the database is already loaded.
  - **DatabaseFile** Ignored if DatabaseName is specified and a database with this name is already running.

The interface library uses the completed list of connection parameters to attempt to connect.

Locating a database server

SQL Anywhere searches for the server name specified in the ServerName (Server) connection parameter. SQL Anywhere searches for a default server if the ServerName (Server) connection parameter is not used, and the CommLinks (LINKS) connection parameter is not specified or if the CommLinks (LINKS) connection parameter is specified and includes Shared Memory.



If SQL Anywhere locates a server, it tries to locate or load the required database on that server. See "Locating the database" on page 144.

If SQL Anywhere can not locate a server, it may attempt to start a personal server, depending on the connection parameters.

Notes

- For local connections, locating a server is simple. For connections over a network, you can use the CommLinks (LINKS) connection parameter to tune the search in many ways by supplying network protocol options.
- You can specify a set of network protocol options for each network protocol in the argument to the CommLinks (LINKS) connection parameter.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- Each attempt to locate a server involves two steps. First, SQL Anywhere looks in the server name cache to see if a server of that name is available (this step is skipped if the value of DoBroadcast is none). Second, it uses the available connection parameters to attempt a connection.

- If the server is started automatically, information from the START, DBF, DBKEY, DBS, DBN, Server, and AutoStop connection parameters are used to construct the options for the automatically started server.

- If the server has an alternate server name, you can only use the alternate server name to connect to the database that specified the alternate server name. You cannot use the alternate server name to connect to any other databases running on that database server. See "-sn dbsrv12 database option" on page 260.

See also

- "-xd dbeng12/dbsrv12 server option" on page 239

Locating a database server using the Broadcast Repeater utility

The Broadcast Repeater utility allows SQL Anywhere clients to find SQL Anywhere database servers running on other subnets and through firewalls where UDP broadcasts normally do not reach, without using the HOST connection parameter or LDAP.

To use the Broadcast Repeater utility

1. Start a DBNS (database name service) process on any computer in a subnet.

2. Start a DBNS process on any computer in a different subnet and pass the computer name or IP address of the first computer as a parameter (using the *address* parameter).

   The two DBNS processes make a TCP/IP connection to each other.

3. The DBNS processes now listen for broadcasts on each of their own subnets. Each DBNS process forwards requests over the TCP/IP connection to the other DBNS process, which re-broadcasts the requests on its subnets and also forwards responses back to the originating DBNS process, which sends them to the original client.

4. Regular SQL Anywhere broadcasts on either of the subnets reach database servers on the remote subnet, and clients are able to connect to database servers on the remote subnet without specifying the HOST parameter.

Any number of DBNS processes can communicate with each other. Each DBNS process connects to every other DBNS that it knows about, and the different DBNS processes share their lists of DBNS processes. For example, suppose you start two DBNS processes, A and B. If you start a third DBNS process, C, in a third subnet, passing the address of B to C, then B tells C about A, and C then connects to A.

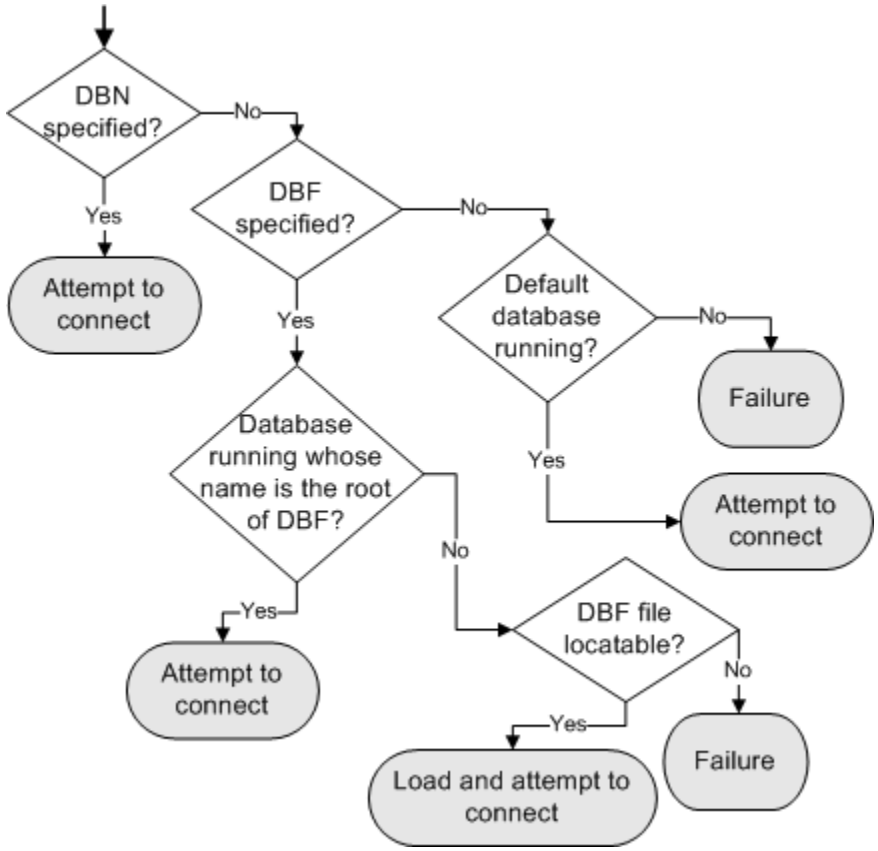
Running more than one DBNS process in a single subnet is not necessary, and is not recommended.

See also

- "Broadcast Repeater utility (dbns12)" on page 772

Locating the database

If SQL Anywhere successfully locates a server, it then tries to locate the database. For example:



Server name caching for faster connections

When the DoBroadcast (DOBROAD) protocol option is set to DIRECT or ALL, the network library looks for a database server on a network by broadcasting over the network using the CommLinks (LINKS) connection parameter.

Tuning the broadcast

The CommLinks (LINKS) parameter takes as an argument a string listing the protocols to use and, optionally for each protocol, a variety of network protocol options that tune the broadcast. See "Network protocol options" on page 311.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Caching server information

Broadcasting over large networks searching for a server of a specific name can be time-consuming. Caching server addresses speeds up network connections by saving the protocol the first connection to a server was found on, and its address, to a file and using that information for subsequent connections.

The server information is saved in a cached file named *sasrv.ini*. The file contains a set of sections, each of the following form:

```
[Server name]
LINKS=protocol_name
Address=address_string
```

The default location of *sasrv.ini* is *%ALLUSERSPROFILE%\Application Data\SQL Anywhere 12* on Windows and *$HOME/.sqlanywhere12* on Unix.

Note
It is very important that each server has a unique name. Giving different servers the same name can lead to identification problems.

How the cache is used

If the server name and protocol in the cache match the connection string, SQL Anywhere tries to connect using the cached address first. If that fails, or if the server name and protocol in the cache do not match the connection string, the connection string information is used to search for the server using a broadcast. If the broadcast is successful, the server name entry in the cache is overwritten. If no server is found, the server name entry in the cache is removed. If the DoBroadcast protocol option is set to none, any cached addresses are ignored.

Interactive SQL connections

Interactive SQL has a different behavior from the default embedded SQL behavior when a CONNECT statement is issued while already connected to a database. If no database or server is specified in the CONNECT statement, Interactive SQL connects to the current database, rather than to the default database. This behavior is required for database reloading operations. See “CONNECT statement [ESQL] [Interactive SQL]” [*SQL Anywhere Server - SQL Reference*].

Testing that a server can be found

Use the dbping utility to troubleshoot connections and determine if a server with a specific name is available on your network.

The dbping utility takes a connection string as an option. The utility does not start the server and only the information required to locate the server are used by default. Use the -d option with the dbping utility to start the server.

Examples

The following command line tests to see if a database server can be found on a computer named Waterloo:

```
dbping -c "Host=Waterloo"
```

The following command tests to see if a default server is available on the current computer.

```
dbping
```

See also

- "Ping utility (dbping)" on page 826

Testing embedded SQL connection performance

You can use the Ping utility (dbping) to obtain information about the performance of embedded SQL connections by specifying the -s or -st options. The following statistics are gathered:

Statistic	Description
DBLib connect and disconnect	The time to perform one DBLib connect and disconnect. Note that the performance of connecting and disconnecting using other interfaces, such as ODBC, is typically slower than DBLib because more requests are required to complete the connection.
Round trip simple request	The time it takes to send a request from the client to the server plus the time it takes to send a response from the server back to the client. The round trip time is twice the average latency.
Send throughput	The throughput based on transferring 100 KB of data for each iteration from dbping to the database server.
Receive throughput	The throughput based on transferring 100 KB of data for each iteration from the database server to dbping.

If your network has both high round trip times and high throughput, the reported throughput will be lower than your actual network throughput because of the high round trip times. Using dbping -s can be useful to give an indication of whether communication compression may improve performance. The performance statistics are approximate, and are more accurate when both the client and server computers are fairly idle. The transferred data can be compressed to approximately 25% of its original size if communication compression is used.

The following is an example of the output from dbping -s for the dbping command `dbping -s -c "UID=DBA;PWD=sql;Host=10.25.107.108;Server=sampleserver;"` :

```
SQL Anywhere Server Ping Utility Version 12.0.0.2413
Connected to SQL Anywhere 12.0.0.2413 server "sampleserver" and database
"sample" at address 10.25.107.108.
Performance statistic           Number            Total Time    Average
----------------------------    --------------    ----------    ------------
DBLib connect and disconnect      175 times       1024 msec         5 msec
Round trip simple request        2050 requests    1024 msec        <1 msec
Send throughput                   7600 KB         1024 msec      7421 KB/sec
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
Receive throughput              10100 KB          1024 msec      9863 KB/sec
Ping database successful.
```

See also

- "Ping utility (dbping)" on page 826

The SQL Anywhere database server

Starts a personal database server or network database server.

Syntax

{ **dbeng12** | **dbsrv12** }
[*server-options*] [*database-file* [*database-options*] ...]

Server options

Server option	Description
@*data*	Reads in options from a configuration file or environment variable. See "@data dbeng12/dbsrv12 server option" on page 157.
-?	Displays usage information. See "-? dbeng12/dbsrv12 server option" on page 158.
-b	Runs in bulk operations mode. See "-b dbeng12/dbsrv12 server option" on page 158.
-c *size*	Sets initial cache size. See "-c dbeng12/dbsrv12 server option" on page 159.
-ca 0	Disables dynamic cache sizing [Windows, Unix, Mac OS X]. See "-ca dbeng12/dbsrv12 server option" on page 161.
-cc{ + | - }	Collects information about database pages to be used for cache warming. See "-cc dbeng12/dbsrv12 server option" on page 162.
-ch *size*	Sets the cache size upper limit [Windows, Unix, Mac OS X]. See "-ch dbeng12/dbsrv12 server option" on page 163.
-chx *size*	Reserves address space for non-cache use [32-bit Windows, 32-bit Unix]. See "-ch dbeng12/dbsrv12 server option" on page 163.
-cl *size*	Sets the cache size lower limit [Windows, Unix, Mac OS X]. See "-cl dbeng12/dbsrv12 server option" on page 166.
-cm *size*	Specifies the amount of address space allocated for an Address Windowing Extensions (AWE) cache [Windows]. See "-cm dbeng12/dbsrv12 server option" on page 167.

Server option	Description
-cp *location*[**;***location ...*]	Specifies set of directories or JAR files in which to search for classes. See "-cp dbeng12/dbsrv12 server option" on page 168.
-cr{ + | - }	Warms the cache with database pages. See "-cr dbeng12/dbsrv12 server option" on page 169.
-cs	Displays cache usage in the database server messages window. See "-cs dbeng12/dbsrv12 server option" on page 169.
-cv{ + | - }	Controls the appearance of messages about cache warming in the database server messages window. See "-cv dbeng12/dbsrv12 server option" on page 170.
-cw	Enables use of Address Windowing Extensions for setting the size of the database server cache [Windows]. See "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171.
-dt *temp-file-dir*	Specifies the directory where temporary files are stored. See "-dt dbeng12/dbsrv12 server option" on page 175.
-ec *encryption-options*	Enables packet encryption [network server]. See "-ec dbeng12/dbsrv12 server option" on page 176.
-ep	Prompts for encryption key. See "-ep dbeng12/dbsrv12 server option" on page 179.
-es	Allows unencrypted connections over shared memory. See "-es dbeng12/dbsrv12 server option" on page 180.
-f	Forces the database to start without a transaction log. See "-f dbeng12/dbsrv12 server recovery option" on page 180.
-fc *filename*	Specifies the file name of a DLL containing the file system full callback function. See "-fc dbeng12/dbsrv12 server option" on page 181.
-fips	Requires the use of FIPS-approved algorithms for strong database and communication encryption [Windows, Unix, and Linux]. See "-fips dbeng12/dbsrv12 server option" on page 182.
-ga	Automatically unloads the database after the last non-HTTP client connection is closed. In addition, shut down after the last database is closed. See "-ga dbeng12/dbsrv12 server option" on page 183.
-gb *level*	Sets database process priority class to *level* [Windows, Unix, Mac OS X]. See "-gb dbeng12/dbsrv12 server option" on page 184.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Server option	Description
-gc *num*	Sets maximum checkpoint timeout period to *num* minutes. See "-gc dbeng12/dbsrv12 server option" on page 184.
-gd *level*	Sets database starting permission. See "-gd dbeng12/dbsrv12 server option" on page 185.
-ge *size*	Sets the stack size for threads that run external functions. See "-ge dbeng12/dbsrv12 server option" on page 186.
-gf	Disables firing of triggers. See "-gf dbeng12/dbsrv12 server option" on page 187.
-gk *level*	Sets the permission required to stop the server. See "-gk dbeng12/dbsrv12 server option" on page 187.
-gl *level*	Sets the permission required to load or unload data. See "-gl dbeng12/dbsrv12 server option" on page 188.
-gm *num*	Sets the maximum number of connections. See "-gm dbeng12/dbsrv12 server option" on page 188.
-gn	Sets the multiprogramming level of the database server. See "-gn dbsrv12 server option" on page 189.
-gna	Controls automatic tuning of the database server multiprogramming level. See "-gna dbsrv12 server option" on page 190.
-gnh *num*	Sets the maximum number of tasks that the database server can execute concurrently. See "-gnh dbsrv12 server option" on page 190.
-gnl	Sets the minimum number of tasks that the database server can execute concurrently. See "-gnl dbsrv12 server option" on page 192.
-gns	Reports multiprogramming level statistics in the database server message log. See "-gns dbsrv12 server option" on page 192.
-gp *size*	Sets the maximum page size to *size* bytes. See "-gp dbeng12/dbsrv12 server option" on page 193.
-gr *minutes*	Sets the maximum recovery time. See "-gr dbeng12/dbsrv12 server option" on page 194.
-gss *size*	Sets the thread stack size to *size* bytes. See "-gss dbeng12/dbsrv12 server option" on page 195.

Server option	Description
-gt *num*	Sets the maximum number of physical processors that can be used (up to the licensed maximum). This option is only useful on multiprocessor systems. See "-gt dbeng12/dbsrv12 server option" on page 195.
-gtc *logical-processors-to-use*	Controls the maximum processor concurrency that the database server allows. See "-gtc dbeng12/dbsrv12 server option" on page 196.
-gu *level*	Sets the permission level for utility commands: utility_db, all, none, or DBA. See "-gu dbeng12/dbsrv12 server option" on page 198.
-im *submode*	Runs the database server in memory, reducing or eliminating writes to disk. See "-im dbeng12/dbsrv12 server option" on page 199.
-k	Controls the collection of Performance Monitor statistics. See "-k dbeng12/dbsrv12 server option" on page 200.
-kl *GSS-API-library-file*	Specifies the file name of the Kerberos GSS-API library (or shared object on Unix) and enable Kerberos authenticated connections to the database server. See "-kl dbeng12/dbsrv12 server option" on page 201.
-kr *server-realm*	Specifies the realm of the Kerberos server principal and enables Kerberos authenticated connections to the database server. See "-kr dbeng12/dbsrv12 server option" on page 202.
-krb	Enables Kerberos-authenticated connections to the database server. See "-krb dbeng12/dbsrv12 server option" on page 203.
-ks	Disables the creation of shared memory that the Performance Monitor uses to collect counter values from the database server [Windows]. See "-ks dbeng12/dbsrv12 server option" on page 204.
-ksc	Specifies the maximum number of connections that the Performance Monitor can monitor [Windows]. See "-ksc dbeng12/dbsrv12 server option" on page 204.
-ksd	Specifies the maximum number of databases that the Performance Monitor can monitor [Windows]. See "-ksd dbeng12/dbsrv12 server option" on page 205.
-m	Truncates the transaction log after each checkpoint for all databases. See "-m dbeng12/dbsrv12 server option" on page 205.
-n *name*	Uses *name* as the name of the database server. Note that the -n option is positional. See "-n dbeng12/dbsrv12 server option" on page 206.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Server option	Description
-o *filename*	Outputs messages to the specified file. See "-o dbeng12/dbsrv12 server option" on page 208.
-oe *filename*	Specifies file to log startup errors, fatal errors and assertions to. See "-oe dbeng12/dbsrv12 server option" on page 209.
-on *size*	Specifies a maximum size for the database server message log file, after which the file is renamed with the extension *.old* and a new file is started. See "-on dbeng12/dbsrv12 server option" on page 209.
-os *size*	Limits the size of the log file for messages. See "-os dbeng12/dbsrv12 server option" on page 210.
-ot *filename*	Truncates the database server message log file and appends output messages to it. See "-ot dbeng12/dbsrv12 server option" on page 211.
-p *packet-size*	Sets the maximum communication packet size [network server]. See "-p dbeng12/dbsrv12 server option" on page 212.
-pc	Compresses all communication packets except same-computer connections. See "-pc dbsrv12 server option" on page 212.
-pt *size-in-bytes*	Sets the minimum network packet size to compress. See "-pt dbsrv12 server option" on page 213.
-qi	Does not display the database server system tray icon or database server messages window [Windows]. See "-qi dbeng12/dbsrv12 server option" on page 214.
-qn	Does not minimize the database server messages window on startup [Windows and Linux]. See "-qn dbeng12/dbsrv12 server option" on page 214.
-qp	Suppresses messages about performance in the database server messages window. See "-qp dbeng12/dbsrv12 server option" on page 215.
-qs	Suppresses startup error windows [Windows]. See "-qs dbeng12/dbsrv12 server option" on page 216.
-qw	Does not display the database server messages window. See "-qw dbeng12/dbsrv12 server option" on page 216.
-r	Opens database in read-only mode. See "-r dbeng12/dbsrv12 server option" on page 217.
-s *facility-ID*	Sets the Syslog facility ID [Unix, Mac OS X]. See "-s dbeng12/dbsrv12 server option" on page 218.

Server option	Description
-sb { **0** | **1** }	Specifies how the server reacts to broadcasts. See "-sb dbeng12/dbsrv12 server option" on page 219.
-sf *feature-list*	Secures features for databases running on this database server. See "-sf dbeng12/dbsrv12 server option" on page 219.
-sk *key*	Specifies a key that can be used to enable features that are disabled for the database server. See "-sk dbeng12/dbsrv12 server option" on page 224.
-su *password*	Sets the password for the DBA user of the utility database (utility_db), or disable connections to the utility database. See "-su dbeng12/dbsrv12 server option" on page 225.
-ti *minutes*	Sets the client idle time before shutdown—default 240 minutes. See "-ti dbeng12/dbsrv12 server option" on page 226.
-tl *seconds*	Sets the default liveness timeout for clients in seconds—default 120 seconds. See "-tl dbeng12/dbsrv12 server option" on page 226.
-tmf	Forces transaction manager recovery for distributed transactions [Windows]. See "-tmf dbeng12/dbsrv12 server option" on page 227.
-tmt *milliseconds*	Sets the re-enlistment timeout for distributed transactions [Windows]. See "-tmt dbeng12/dbsrv12 server option" on page 228.
-tq *time*	Sets quitting time [network server]. See "-tq dbeng12/dbsrv12 server option" on page 228.
-u	Uses buffered disk I/O [Windows, Unix, Mac OS X]. See "-u dbeng12/dbsrv12 server option" on page 229.
-ua	Turns off use of asynchronous I/O [Linux]. See "-ua dbeng12/dbsrv12 server option" on page 229.
-uc	Starts the database server in shell mode [Unix and Mac OS X]. See "-uc dbeng12/dbsrv12 server option" on page 230.
-ud	Runs as a daemon [Unix, Mac OS X]. See "-ud dbeng12/dbsrv12 server option" on page 230.
-uf	Specifies the action to take when a fatal error occurs [Unix, Mac OS X]. See "-uf dbeng12/dbsrv12 server option" on page 231.
-ui	Opens the **Server Startup Options** window and displays the database server messages window, or starts the database server in shell mode if a usable display isn't available [Linux and Mac OS X]. See "-ui dbeng12/dbsrv12 server option" on page 232.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Server option	Description
-um	Opens the **Server Startup Options** window and displays the database server messages window [Mac OS X]. See "-um dbeng12/dbsrv12 server option" on page 232.
-ut *minutes*	Touches temporary files every *min* minutes [Unix, Mac OS X]. See "-ut dbeng12/dbsrv12 server option" on page 233.
-ux	Displays the database server messages window and **Server Startup Options** window [Linux]. See "-ux dbeng12/dbsrv12 server option" on page 233.
-v	Displays database server version and stop. See "-v dbeng12/dbsrv12 server option" on page 234.
-vss{ + | - }	Enables and disables the Volume Shadow Copy Service (VSS) [Windows]. See "-vss dbeng12/dbsrv12 server option" on page 235.
-wc[+ | -]	Enables write checksums for databases running on the database server. See "-wc dbeng12/dbsrv12 server option" on page 235.
-x *list*	Specifies a comma-separated list of communication protocols to use. See "-x dbeng12/dbsrv12 server option" on page 236.
-xa *authentication-info*	Specifies a list of database names and authentication strings for an arbiter server. See "-xa dbsrv12 server option" on page 238.
-xd	Prevents the database server from becoming the default database server. See "-xd dbeng12/dbsrv12 server option" on page 239.
-xf *state-file*	Specifies the location of the file used for maintaining state information about your database mirroring system. See "-xf dbsrv12 server option" on page 239.
-xm *seconds*	Sets the time to check for new IP addresses in seconds. The minimum value is 10 and the default value is 0. For a portable device, the default value is 120. See "-xm dbeng12/dbsrv12 server option" on page 240.
-xs	Specifies server side web services communications protocols. See "-xs dbeng12/dbsrv12 server option" on page 241.
-z	Provides diagnostic information on communication links [network server]. See "-z dbeng12/dbsrv12 server option" on page 243.
-ze	Displays database server environment variables in the database server messages window. See "-ze dbeng12/dbsrv12 server option" on page 243.

Server option	Description
-zl	Turns on capturing of the most recently-prepared SQL statement for each connection. See "-zl dbeng12/dbsrv12 server option" on page 244.
-zn *integer*	Specifies the number of request log file copies to retain. See "-zn dbeng12/dbsrv12 server option" on page 245.
-zo *filename*	Redirects request logging information to a separate file. See "-zo dbeng12/dbsrv12 server option" on page 246.
-zoc	Redirects web service client information to a file. See "-zoc dbeng12/dbsrv12 server option" on page 246.
-zp	Turns on capturing of the plan most recently used by the query optimizer. See "-zp dbeng12/dbsrv12 server option" on page 247.
-zr { **all** | **SQL** | **none** }	Turns on logging of SQL operations. The default is NONE. See "-zr dbeng12/dbsrv12 server option" on page 248.
-zs *size*	Limits the size of the log file used for request logging. See "-zs dbeng12/dbsrv12 server option" on page 249.
-zt	Turns on logging of request timing information. See "-zt dbeng12/dbsrv12 server option" on page 250.

Database options

The following options can only be specified after a database file name in the database server command.

Database option	Description
-a *filename*	Applies the named transaction log file. See "-a dbeng12/dbsrv12 database option" on page 251.
-ad *log-directory*	Specifies the directory containing transaction log files to be applied to the database. See "-ad dbeng12/dbsrv12 database option" on page 252.
-ar	Applies any log files located in the same directory as the transaction log to the database. See "-ar dbeng12/dbsrv12 database option" on page 253.
-as	Continues running the database after transaction logs have been applied (used in conjunction with -ad or -ar). See "-as dbeng12/dbsrv12 database option" on page 253.
-dh	Does not display the database when dblocate is used against this server. See "-dh dbeng12/dbsrv12 database option" on page 255.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Database option	Description
-ds	Specifies the location of the dbspaces for the database. See "-ds dbeng12/dbsrv12 database option" on page 254.
-ek *key*	Specifies encryption key. See "-ek dbeng12/dbsrv12 database option" on page 256.
-m	Truncates (deletes) the transaction log after each checkpoint for the specified database. See "-m dbeng12/dbsrv12 database option" on page 256.
-n *name*	Names the database. See "-n dbeng12/dbsrv12 database option" on page 257.
-r	Opens the specified database(s) in read-only mode. Database modifications not allowed. See "-r dbeng12/dbsrv12 database option" on page 258.
-sm	Provides a database server name that can be used to access the read-only mirror database. See "-sm dbsrv12 database option (deprecated)" on page 259.
-sn *alternate-server-name*	Provides an alternate server name for a single database running on a database server. See "-sn dbsrv12 database option" on page 260.
-wc[+ | -]	Enables write checksums for databases running on the database server. See "-wc dbeng12/dbsrv12 database option" on page 262.
-xp *mirroring-options*	Provides information to an operational server that allows it to connect to its partner and to the arbiter when database mirroring is being used. See "-xp dbsrv12 database option" on page 263.

Remarks

The elements of the database server command include the following:

- **Executable** The **dbeng12** command starts a personal database server. The **dbsrv12** command starts a network database server.

  Both personal and network database servers are supplied for each supported operating system, with one exception. On Windows Mobile, only the network database server is supplied. The support for TCP/IP in the network server enables you to perform tasks from your desktop computer, including database management with Sybase Central.

  On Windows operating systems, except Windows Mobile, the name of the personal database server executable is *dbeng12.exe*. On Unix operating systems its name is dbeng12.

  On Windows operating systems, including Windows Mobile, the name of the network database server executable is *dbsrv12.exe*. On Linux and Unix operating systems, the name is dbsrv12.

- **Server options** These options control the behavior of the database server for all running databases.

- **Database file** You can specify zero, one, or more database file names. Each of these databases starts and remains available for applications.

  Caution
  The database file and the transaction log file must be located on the same physical computer as the database server or accessed via a SAN or iSCSI configuration. Database files and transaction log files located on a remote network directory can lead to poor performance, data corruption, and database server instability.

  For more information, see http://www.sybase.com/detail?id=1034790.

  For best results, the transaction log should be kept on a different disk from the database files. See "The transaction log" on page 21.

- **Database options** For each database file you start, you can provide database options that control certain aspects of its behavior.

Database and server options are generally case sensitive. You should enter all options in lowercase.

The *database-file* specifies the database file name. If *database-file* is specified without a file extension, SQL Anywhere looks for *database-file* with extension *.db*. If you use a relative path, it is read relative to the current working directory. You can supply a full path.

If you supply no options and no database file, then on Windows operating systems a window appears, allowing you to browse to your database file.

If you want to start a database server from a batch file, you must use the dbspawn utility. See "Start Server in Background utility (dbspawn)" on page 849.

The personal database server has a maximum of ten concurrent connections, uses at most one CPU for request processing, and doesn't support network client/server connections. By default, the personal database server only uses the shared memory protocol. You must use the -x option if you want to use TCP/IP with the personal database server. See "-x dbeng12/dbsrv12 server option" on page 236.

In addition, there are other minor differences, such as the default permission level that is required to start new databases, or the permissions required to execute the CHECKPOINT statement. For more information about the differences between the personal database server and the network database server, see "Using SQL Anywhere database servers" on page 33.

By default, the database server page size is the same as the largest page size of the databases on the command line. Once the database server starts, you cannot start a database with a larger page size than the database server. See "Setting a maximum page size" on page 39.

Example

The following command starts the SQL Anywhere sample database running on a personal database server:

```
dbeng12 "%SQLANYSAMP12%\demo.db"
```

The following command starts the SQL Anywhere sample database running on a network database server:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
dbsrv12 "%SQLANYSAMP12%\demo.db"
```

The following example, entered all on one line, starts a database server named **myserver** that starts with a cache size of 3 MB and loads the sample database:

```
dbeng12 -c 3m -n myserver "%SQLANYSAMP12%\demo.db"
```

For more examples showing how to start a database server, see "Examples of starting a database server" on page 34.

Database server options

These options apply to the database server as a whole, not just to an individual database.

@data dbeng12/dbsrv12 server option

Reads in options from the specified environment variable or configuration file.

Syntax

{ **dbeng12** | **dbsrv12** } *@data ...*

Applies to

All operating systems and database servers, except Windows Mobile. It is supported for all database utilities except the Language Selection utility (dblang), the Certificate Creation utility (createcert), the Certificate Viewer utility (viewcert), the Microsoft ActiveSync provider install utility (mlasinst), and the File Hiding utility (dbfhide).

Remarks

Use this option to read in command line options from the specified environment variable or configuration file. If both exist with the same name that is specified, the environment variable is used.

Configuration files can contain line breaks, and can contain any set of options. See "Using configuration files" on page 763.

If you want to protect the information in a configuration file (for example, because it contains passwords) you can use the File Hiding (dbfhide) utility to obfuscate the contents of configuration files. See "File Hiding utility (dbfhide)" on page 794.

The @data parameter can occur at any point in the command line, and parameters contained in the file are inserted at that point. Multiple files can be specified, and the file specifier can be used with command line options.

See also

- "Using configuration files" on page 763

Example

The following configuration file holds a set of options for a server named **myserver** that starts with a cache size of 4 MB and loads the sample database:

```
-c 4096
-n myserver
"c:\mydatabase.db"
```

If this configuration file is saved as *c:\config.txt*, it can be used in a command as follows:

```
dbsrv12 @c:\config.txt
```

The following configuration file contains comments:

```
#This is the server name:
-n MyServer
#These are the protocols:
-x tcpip
#This is the database file
my.db
```

The following statement sets an environment variable that holds options for a database server that starts with a cache size of 4 MB and loads the sample database.

```
SET envvar=-c 4096 "c:\mydatabase.db";
```

This command starts the database server using an environment variable named **envvar**.

```
dbsrv12 @envvar
```

-? dbeng12/dbsrv12 server option

Displays usage information.

Syntax

{ **dbeng12** | **dbsrv12** } **-?**

Applies to

All operating systems and database servers, except Windows Mobile.

Remarks

When you specify this option, a brief description of each server option appears.

-b dbeng12/dbsrv12 server option

Uses bulk operation mode.

Syntax

{ **dbeng12** | **dbsrv12** } **-b** ...

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Applies to

All operating systems and database servers.

Remarks

This option is useful for using the Interactive SQL INPUT statement to load large quantities of data into a database. See "INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*].

The -b option should not be used if you are using LOAD TABLE to bulk load data. See "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*].

When you use this option, the database server allows only one connection by one application. It keeps a rollback log, but it doesn't keep a transaction log. The multi-user locking mechanism is turned off.

When you first start the database server after loading data with the -b option, you should use a new transaction log file.

Bulk operation mode doesn't disable the firing of triggers.

See also

- "Data recovery issues for bulk operations" [*SQL Anywhere Server - SQL Usage*]
- "Performance aspects of bulk operations" [*SQL Anywhere Server - SQL Usage*]

-c dbeng12/dbsrv12 server option

Sets the initial memory reserved for caching database pages and other database server information.

Syntax

{ **dbeng12** | **dbsrv12** } **-c** { *size*[**k** | **m** | **g** | **p**] } ...

Applies to

All operating systems and database servers.

Remarks

The amount of memory available for use as a database server cache is one of the key factors controlling performance. You can set the initial amount of cache memory using the -c server option. The more cache memory that can be given to the database server, the better its performance.

The *size* is the amount of memory, in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively.

The unit **p** is a percentage either of the total physical system memory, or of the maximum non-AWE cache size, whichever is lower. The maximum non-AWE cache size depends on the operating system. For example:

- 2.5 GB for Windows 32-bit Advanced Server, Enterprise Server, Datacenter Server, and Vista

- 3.5 GB for the 32-bit database server running on Windows x64 Edition
- 1.5 GB on all other 32-bit Windows operating systems
- On Windows Mobile, the cache size is limited by available physical memory
- On 64-bit database servers, the cache size can be considered unlimited

If you use **p**, the argument is a percentage. You can use % as an alternative to **p**, but on Windows operating systems, which use % as an environment variable escape character, you must escape the % character. To set the initial cache size to 50 percent of the physical system memory, you would run the following command:

```
dbeng12 -c 50%% ...
```

If no -c option is provided, the database server calculates the initial cache allocation as follows:

- **Windows Mobile** The formula is as follows:

  ```
  max( 600 KB, min( dbsize, 0.25*TotalPhysicalMemory ) );
  ```

  The *dbsize* is the total size of the database file or files started, and *TotalPhysicalMemory* the total amount of physical memory on the computer.

- **Windows** The formula is as follows:

  ```
  max( 2 MB, min( dbsize, 0.25*TotalPhysicalMemory ) );
  ```

  The *dbsize* is the total size of the database file or files started, and *TotalPhysicalMemory* is the total amount of physical memory on the computer.

  If an AWE cache is used on Windows the formula is as follows:

  ```
  min( 100% of AvailablePhysicalMemory-128MB, dbsize );
  ```

  An AWE cache is not used if this value is smaller than 2 MB.

  For information about AWE caches, see "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171.

  Note
  The use of AWE is deprecated. It is recommended that you use the 64-bit version of the SQL Anywhere database server on a 64-bit Windows operating system if you require a large cache.

- **Unix** At least 8 MB.

  ```
  max( 8 MB, min( 0.1*( physical-memory + available-swap ) , databbase-
  size*1.1 )
  ```

  For information about Unix initial cache size, see "Dynamic cache sizing on Unix" [*SQL Anywhere Server - SQL Usage*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

If you disable dynamic cache resizing (-ca option), then the cache size that is used may be restricted by the amount of memory that is available. See "Use the cache to improve performance" [*SQL Anywhere Server - SQL Usage*].

The database server messages window displays the size of the cache at startup and you can use the following statement to obtain the current size of the cache:

```
SELECT PROPERTY( 'CurrentCacheSize' );
```

See also

- "Cache size" on page 44
- "Dynamic cache sizing" [*SQL Anywhere Server - SQL Usage*]
- "-ca dbeng12/dbsrv12 server option" on page 161
- "-cc dbeng12/dbsrv12 server option" on page 162
- "-ch dbeng12/dbsrv12 server option" on page 163
- "-chx dbeng12/dbsrv12 server option" on page 164
- "-cl dbeng12/dbsrv12 server option" on page 166
- "-cm dbeng12/dbsrv12 server option" on page 167
- "-cr dbeng12/dbsrv12 server option" on page 169
- "-cs dbeng12/dbsrv12 server option" on page 169
- "-cv dbeng12/dbsrv12 server option" on page 170
- "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171
- "Use the cache to improve performance" [*SQL Anywhere Server - SQL Usage*]

Example

The following example, entered all on one line, starts a database server named **myserver** that starts with a cache size of 3 MB and loads the sample database:

```
dbeng12 -c 3m -n myserver "samples-dir\demo.db"
```

For information about *samples-dir*, see "Samples directory" on page 392.

-ca dbeng12/dbsrv12 server option

Enforces a static cache size.

Syntax

{ **dbeng12** | **dbsrv12** } **-ca 0** ...

Applies to

Windows, Unix, Mac OS X

Remarks

You can disable automatic cache size tuning by specifying -ca 0 on the command line. If you do not include the -ca 0 option, the database server automatically increases the cache size. If you specify this option, the cache size is still adjusted if the database server would otherwise run into an error indicating that the dynamic memory is exhausted.

This server option must only be used in the form -ca 0.

This option is ignored if you are using an AWE cache. See "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171.

Note
The use of AWE is deprecated. It is recommended that you use the 64-bit version of the SQL Anywhere database server on a 64-bit Windows operating system if you require a large cache.

See also

- "Cache size" on page 44
- "Dynamic cache sizing" [*SQL Anywhere Server - SQL Usage*]
- "-c dbeng12/dbsrv12 server option" on page 159
- "-cc dbeng12/dbsrv12 server option" on page 162
- "-ch dbeng12/dbsrv12 server option" on page 163
- "-chx dbeng12/dbsrv12 server option" on page 164
- "-cl dbeng12/dbsrv12 server option" on page 166
- "-cm dbeng12/dbsrv12 server option" on page 167
- "-cr dbeng12/dbsrv12 server option" on page 169
- "-cs dbeng12/dbsrv12 server option" on page 169
- "-cv dbeng12/dbsrv12 server option" on page 170
- "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171

Example

The following example starts a database server named myserver that has a static cache that is 40% of the total physical memory and loads the sample database, and the database server does not automatically adjust the cache size:

```
dbsrv12 -c 40P -ca 0 -n myserver "samples-dir\demo.db"
```

For information about *samples-dir*, see "Samples directory" on page 392.

-cc dbeng12/dbsrv12 server option

Collects information about database pages to be used for cache warming the next time the database is started.

Syntax

{ **dbeng12** | **dbsrv12** } **-cc**{ **+** | **-** } ...

Applies to

All operating systems and database servers.

Remarks

By default, page collection is turned on. When collection is turned on, the database server keeps track of each database page that is requested. Collection stops when the maximum number of pages has been collected, the database is shut down, or the collection rate falls below the minimum value. You cannot

configure the maximum number of pages collected or specify the value for the collection rate (the value is based on cache size and database size). Once collection stops, information about the requested pages is recorded in the database so those pages can be used to warm the cache the next time the database is started with the -cr option. Collection of referenced pages is turned on by default.

See also

- "-c dbeng12/dbsrv12 server option" on page 159
- "-ca dbeng12/dbsrv12 server option" on page 161
- "-ch dbeng12/dbsrv12 server option" on page 163
- "-chx dbeng12/dbsrv12 server option" on page 164
- "-cl dbeng12/dbsrv12 server option" on page 166
- "-cm dbeng12/dbsrv12 server option" on page 167
- "-cr dbeng12/dbsrv12 server option" on page 169
- "-cs dbeng12/dbsrv12 server option" on page 169
- "-cv dbeng12/dbsrv12 server option" on page 170
- "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171
- "Using cache warming" [*SQL Anywhere Server - SQL Usage*]

-ch dbeng12/dbsrv12 server option

Sets a maximum cache size, as a limit to automatic cache growth.

Syntax

{ **dbeng12** | **dbsrv12** } **-ch** { *size*[**k** | **m** | **g** | **p**] } ...

Applies to

Windows, Unix, Mac OS X

Remarks

This option limits the size of the database server cache during automatic cache growth. By default the upper limit is approximately the lower of the maximum non-AWE cache size and 90% of the total physical memory of the computer.

The *size* is the amount of memory, in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively.

The unit **p** is a percentage either of the physical system memory, or of the maximum non-AWE cache size, whichever is lower. The maximum non-AWE cache size depends on the operating system. For example:

- 2.5 GB for Windows 32-bit Advanced Server, Enterprise Server and Datacenter Server
- 3.5 GB for the 32-bit database server running on Windows x64 Edition
- 1.5 GB on all other 32-bit Windows operating systems
- On Windows Mobile, the cache size is limited by available physical memory

- On 64-bit database servers, the cache size can be considered unlimited

If you use **p**, the argument is a percentage. You can use % as an alternative to **p**, but on Windows operating systems, which use % as an environment variable escape character, you must escape the % character. To set the initial cache size to 50 percent of the physical system memory, you would run the following command:

On 64-bit operating systems, -ch and -chx are equivalent, but -ch is recommended.

```
dbeng12 -ch 50%% ...
```

This option is ignored if you are using an AWE cache. See "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171.

See also

- "Cache size" on page 44
- "Dynamic cache sizing" [*SQL Anywhere Server - SQL Usage*]
- "-c dbeng12/dbsrv12 server option" on page 159
- "-ca dbeng12/dbsrv12 server option" on page 161
- "-cc dbeng12/dbsrv12 server option" on page 162
- "-chx dbeng12/dbsrv12 server option" on page 164
- "-cl dbeng12/dbsrv12 server option" on page 166
- "-cm dbeng12/dbsrv12 server option" on page 167
- "-cr dbeng12/dbsrv12 server option" on page 169
- "-cs dbeng12/dbsrv12 server option" on page 169
- "-cv dbeng12/dbsrv12 server option" on page 170

Example

The following example starts a database server named silver that has a maximum cache size of 2 MB and loads the sample database:

```
dbeng12 -ch 2m -n silver "samples-dir\demo.db"
```

For information about *samples-dir*, see "Samples directory" on page 392.

-chx dbeng12/dbsrv12 server option

Sets a maximum cache size as a limit to automatic cache growth on 32-bit database servers.

Syntax

{ **dbeng12** | **dbsrv12** } **-chx** { *size*[**k** | **m** | **g** | **p**] } ...

Default

Minimum of 512 MB of address space is reserved for non-cache use

Applies to

32-bit Windows and 32-bit Unix

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

Caution
You should only use the -chx option if you have performed testing to ensure that using additional address space does not affect other components that use server address space such as DLLs that the server must load, ODBC drivers for remote table access, network packet buffers, external stored procedures, and non-cache memory allocations.

If you need a large cache, it is recommended that you use the 64-bit version of the SQL Anywhere database server on a 64-bit operating system.

The database server normally limits the cache size so that at least 512 MB of address space is reserved for use outside the cache. If you want to specify a maximum cache size that reserves less address space for non-cache use the -chx option. This option applies only to 32-bit database servers. Using larger cache sizes may lead to database server instability. This option should be used with caution.

The maximum non-AWE cache size depends on the operating system. For example:

- 2.5 GB for Windows 32-bit Advanced Server, Enterprise Server, Datacenter Server, and Vista
- 3.5 GB for the 32-bit database server running on Windows x64 Edition
- 1.5 GB on all other 32-bit Windows operating systems
- On Windows Mobile, the cache size is limited by available physical memory
- On 64-bit database servers, the cache size can be considered unlimited

On 64-bit operating systems, -ch and -chx are equivalent, but -ch is recommended.

The *size* is the amount of memory, in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively.

The unit **p** is a percentage either of the physical system memory, or of the maximum non-AWE cache size, whichever is lower. You can use % as an alternative to **p**, but on Windows, which uses % as an environment variable escape character, you must escape the % character.

See also

- "Cache size" on page 44
- "-c dbeng12/dbsrv12 server option" on page 159
- "-ca dbeng12/dbsrv12 server option" on page 161
- "-cc dbeng12/dbsrv12 server option" on page 162
- "-ch dbeng12/dbsrv12 server option" on page 163
- "-cl dbeng12/dbsrv12 server option" on page 166
- "-cm dbeng12/dbsrv12 server option" on page 167
- "-cr dbeng12/dbsrv12 server option" on page 169
- "-cs dbeng12/dbsrv12 server option" on page 169
- "-cv dbeng12/dbsrv12 server option" on page 170
- "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171
- "Use the cache to improve performance" [*SQL Anywhere Server - SQL Usage*]

-cl dbeng12/dbsrv12 server option

Sets a minimum cache size as a lower limit to dynamic cache resizing.

Syntax

{ **dbeng12** | **dbsrv12** } **-cl** { *size*[**k** | **m** | **g** | **p**] } ...

Default

2 MB on Windows

8 MB on Unix

Applies to

Windows, Unix, Mac OS X

Remarks

This option sets a lower limit to the cache. If -c is specified, and -cl is not specified, then the minimum cache size is set to the initial cache size (the -c setting). If neither -c nor -cl is set, the minimum cache set is set to a low, constant value, so that the cache can shrink if necessary. On Windows platforms this value is 2 MB.

The *size* is the amount of memory, in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively.

The unit **p** is a percentage either of the physical system memory, or of the maximum non-AWE cache size, whichever is lower. The maximum non-AWE cache size depends on the operating system. For example:

- 2.5 GB for Windows 32-bit Advanced Server, Enterprise Server and Datacenter Server
- 3.5 GB for the 32-bit database server running on Windows x64 Edition
- 1.5 GB on all other 32-bit systems

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- On Windows Mobile, the cache size is limited by available physical memory
- On 64-bit database servers, the cache size can be considered unlimited

If you use **p**, the argument is a percentage. You can use % as an alternative to **p**, but on Windows, which uses % as an environment variable escape character, you must escape the % character. To set the minimum cache size to 50 percent of the physical system memory, you would use the following:

```
dbeng12 -cl 50%% ...
```

This option is ignored if you are using an AWE cache. See "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171.

See also

- "Dynamic cache sizing" [*SQL Anywhere Server - SQL Usage*]
- "-c dbeng12/dbsrv12 server option" on page 159
- "-ca dbeng12/dbsrv12 server option" on page 161
- "-cc dbeng12/dbsrv12 server option" on page 162
- "-ch dbeng12/dbsrv12 server option" on page 163
- "-chx dbeng12/dbsrv12 server option" on page 164
- "-cm dbeng12/dbsrv12 server option" on page 167
- "-cr dbeng12/dbsrv12 server option" on page 169
- "-cs dbeng12/dbsrv12 server option" on page 169
- "-cv dbeng12/dbsrv12 server option" on page 170
- "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171

Example

The following example starts a database server named silver that has a minimum cache size of 5 MB and loads the database file *example.db*:

```
dbeng12 -cl 5m -n silver "c:\example.db"
```

-cm dbeng12/dbsrv12 server option

Specifies the amount of address space allocated for an Address Windowing Extensions (AWE) cache on Windows.

> **Note**
> The use of AWE is deprecated. It is recommended that you use the 64-bit version of the SQL Anywhere database server on a 64-bit Windows operating system if you require a large cache.

Syntax

{ **dbeng12** | **dbsrv12** } **-cm** { *size*[**k** | **m** | **g** | **p**] } ...

Applies to

Windows

Remarks

When using an AWE cache on any of the supported platforms, the database server uses its entire address space except for 512 MB to access the cache memory. The 512 MB address space is left available for other purposes, such as DLLs that the server must load and for non-cache memory allocations. On most systems, the default setting is enough. If you need to increase or decrease the amount of reserved address space, you can do so by specifying the -cm option. The database server displays the amount of address space it is using in the database server messages window at startup.

The *size* is the amount of memory, in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively.

The unit **p** is a percentage of the maximum non-AWE cache size. If you use **p**, the argument is a percentage. You can use % as an alternative to **p**, but as most Windows operating systems use % as an environment variable escape character, you must escape the % character. To set the AWE address space cache size to 50 percent of the address space available to the database server process, you would use the following:

```
dbeng12 -cm 50%% ...
```

See also

- "-c dbeng12/dbsrv12 server option" on page 159
- "-ca dbeng12/dbsrv12 server option" on page 161
- "-cc dbeng12/dbsrv12 server option" on page 162
- "-ch dbeng12/dbsrv12 server option" on page 163
- "-chx dbeng12/dbsrv12 server option" on page 164
- "-cl dbeng12/dbsrv12 server option" on page 166
- "-cr dbeng12/dbsrv12 server option" on page 169
- "-cs dbeng12/dbsrv12 server option" on page 169
- "-cv dbeng12/dbsrv12 server option" on page 170
- "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171

-cp dbeng12/dbsrv12 server option

Specifies set of directories or JAR files in which to search for classes.

Syntax

{ **dbeng12** | **dbsrv12** } **-cp** *location*[**;***location* ...] ...

Applies to

All operating systems and database servers.

Remarks

It is recommended that all classes and JAR files that are being used with Java in the database be installed in the database. When you store the classes and JAR files within the database, the database can be easily moved to a different computer or operating system. Another benefit of installing classes and JAR files into the database is that the SQL Anywhere class loader can fetch the classes and resources from the

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

database, allowing each connection that is using Java in the database to have its own instance of these classes and its own copy of static variables within these classes.

However, when the class or JAR file must be loaded by the system class loader, it can be specified with the -cp server option. This option adds directories and JAR files to the classpath that the database server builds for launching the Java VM.

See also

- "Java in the database" [*SQL Anywhere Server - Programming*]
- "How do I store Java classes in the database?" [*SQL Anywhere Server - Programming*]

-cr dbeng12/dbsrv12 server option

Reloads (warms) the cache with database pages using information collected the last time the database was run.

Syntax

{ **dbeng12** | **dbsrv12** } **-cr**{ **+** | **-** } ...

Applies to

All operating systems and database servers.

Remarks

You can instruct the database server to warm the cache using pages that were referenced the last time the database was started (page collection is turned on using the -cc option). Cache warming is turned on by default. When a database is started, the server checks the database to see if it contains a collection of pages requested the last time the database was started. If the database contains this information, the previously-referenced pages are then loaded into the cache.

Warming the cache with pages that were referenced the last time the database was started can improve performance when the same query or similar queries are executed against a database each time it is started.

See also

- "-cc dbeng12/dbsrv12 server option" on page 162
- "-cl dbeng12/dbsrv12 server option" on page 166
- "-cm dbeng12/dbsrv12 server option" on page 167
- "-cs dbeng12/dbsrv12 server option" on page 169
- "-cv dbeng12/dbsrv12 server option" on page 170
- "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171
- "Using cache warming" [*SQL Anywhere Server - SQL Usage*]

-cs dbeng12/dbsrv12 server option

Displays statistics related to dynamic cache sizing in the database server messages window.

Syntax

{ **dbeng12** | **dbsrv12** } **-cs** ...

Applies to

Windows, Unix

Remarks

For troubleshooting purposes, this option displays statistics in the database server messages window that database server is using to determine how to tune size of the cache.

See also

- "CacheSizingStatistics server property" on page 646
- "-c dbeng12/dbsrv12 server option" on page 159
- "-ca dbeng12/dbsrv12 server option" on page 161
- "-cc dbeng12/dbsrv12 server option" on page 162
- "-ch dbeng12/dbsrv12 server option" on page 163
- "-chx dbeng12/dbsrv12 server option" on page 164
- "-cl dbeng12/dbsrv12 server option" on page 166
- "-cm dbeng12/dbsrv12 server option" on page 167
- "-cr dbeng12/dbsrv12 server option" on page 169
- "-cv dbeng12/dbsrv12 server option" on page 170
- "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Using cache warming" [*SQL Anywhere Server - SQL Usage*]

-cv dbeng12/dbsrv12 server option

Controls the appearance of messages about cache warming in the database server messages window.

Syntax

{ **dbeng12** | **dbsrv12** } **-cv**{ **+** | **-** } ...

Default

Cache warming messages are suppressed.

Applies to

All operating systems and database servers.

Remarks

When -cv+ is specified, a message appears in the database server messages window when any of the following cache warming activities occur:

- collection of requested pages starts or stops (controlled by the -cc server option)
- page reloading starts or stops (controlled by the -cr server option)

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "-c dbeng12/dbsrv12 server option" on page 159
- "-ca dbeng12/dbsrv12 server option" on page 161
- "-cc dbeng12/dbsrv12 server option" on page 162
- "-ch dbeng12/dbsrv12 server option" on page 163
- "-cl dbeng12/dbsrv12 server option" on page 166
- "-cm dbeng12/dbsrv12 server option" on page 167
- "-cr dbeng12/dbsrv12 server option" on page 169
- "-cs dbeng12/dbsrv12 server option" on page 169
- "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171
- "Using cache warming" [*SQL Anywhere Server - SQL Usage*]

Example

The following command starts the database *mydatabase.db* with database page collection and page loading turned on, and logs messages about these activities to the database server messages window:

```
dbsrv12 -cc+ -cr+ -cv+ mydatabase.db
```

-cw dbeng12/dbsrv12 server option (deprecated)

Enables use of Address Windowing Extensions (AWE) on Windows for setting the size of the database server cache.

Note
The use of AWE is deprecated. It is recommended that you use the 64-bit version of the SQL Anywhere database server on a 64-bit Windows operating system if you require a large cache.

Syntax

{ **dbeng12** | **dbsrv12** } **-cw** ...

Applies to

Windows

Remarks

The amount of memory available for use as a database server cache is one of the key factors controlling performance. Because Windows supports Address Windowing Extensions, you can use the -cw option to take advantage of large cache sizes based on the maximum amount of physical memory in the system.

AWE caches are not supported on 64-bit SQL Anywhere database servers.

Operating system (32-bit)	Maximum non-AWE cache size	Maximum amount of physical memory supported by Windows
Windows XP Home Edition	1.5 GB	2 GB

Operating system (32-bit)	Maximum non-AWE cache size	Maximum amount of physical memory supported by Windows
Windows XP Professional	1.5 GB	4 GB
Windows Server 2003 Web	1.5 GB	2 GB
Windows Server 2003 Standard	1.5 GB	4 GB
Windows Server 2003 Enterprise	2.5 GB[1]	32 GB
Windows Server 2003 Datacenter	2.5 GB[1]	64 GB
Windows Vista Ultimate	2.5 GB[1]	4 GB
Windows Vista Enterprise	2.5 GB[1]	4 GB
Windows Vista Business	2.5 GB[1]	4 GB
Windows Vista Home Premium	2.5 GB[1]	4 GB
Windows Vista Home Basic	2.5 GB[1]	4 GB
Windows Vista Starter	2.5 GB[1]	1 GB
Windows 7 Ultimate	2.5 GB[1]	4 GB
Windows 7 Enterprise	2.5 GB[1]	4 GB
Windows 7 Professional	2.5 GB[1]	4 GB
Windows 7 Home Premium	2.5 GB[1]	4 GB
Windows 7 Home Basic	2.5 GB[1]	4 GB
Windows 7 Starter	2.5 GB[1]	1 GB
Windows 2008 Datacenter	2.5 GB[1]	64 GB
Windows Server 2008 Enterprise	2.5 GB[1]	64 GB
Windows Server 2008 Standard	2.5 GB[1]	4 GB
Windows Server 2008 Small Business Server	2.5 GB[1]	4 GB
Windows Web Server 2008	2.5 GB[1]	4 GB

[1] If you have 2 GB-16 GB of memory, enable 4 gigabyte tuning (4GT):

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Windows XP/2003** Add the **/3GB** option to the Windows boot line in the "[operating systems]" section of the *boot.ini* file.
- **Windows Vista and later** Run the following command as an administrator:

```
bcdedit /set increaseuserva 3072
```

When using an AWE cache, most of the available physical memory in the system can be allocated for the cache.

If you can set a cache of the required size using a non-AWE cache, this is recommended because AWE caches allocate memory that can only be used by the database server. This means that while the database server is running, the operating system and other applications cannot use the memory allocated for the database server cache. AWE caches do not support dynamic cache sizing. Therefore, if an AWE cache is used and you specify the -ch or -cl options to set the upper and lower cache size, they are ignored.

By default, 512 MB of address space is reserved for purposes other than the SQL Anywhere AWE cache (address space is the amount of memory that can be accessed by a program at any given time). Although this amount is usually enough, you can use the -cm option to change the amount of reserved address space.

On Windows Vista and Windows 7, only elevated database servers can use AWE memory. If you are starting a database server automatically on Windows Vista or Windows 7, you must specify ELEVATE=YES in your connection string so that the database server executables are elevated. See "Elevate connection parameter" on page 286.

To start a database server with an AWE cache, you must do the following:

- On Windows Vista and Windows 7, you must run the database server as an administrator.
- Have at least 130 MB of memory available on your system.
- If you have 2 GB-16 GB of memory, enable 4 gigabyte tuning (4GT):
  - **Windows XP/2003** Add the **/3GB** option to the Windows boot line in the "[operating systems]" section of the *boot.ini* file.
  - **Windows Vista and later** Run the following command as an administrator:

```
bcdedit /set increaseuserva 3072
```

- If you have more than 16 GB of memory, do not enable 4GT.
- If you have at least 4 GB of memory, enable PAE:
  - **Windows XP/2003** Add the **/PAE** option to the Windows boot line in the [operating systems] section of the *boot.ini* file.
  - **Windows Vista and later** Run the following command as an administrator:

```
bcdedit /set pae ForceEnable
```

- Grant the Lock Pages in Memory privilege to the user ID under which the server is run. The following steps explain how to do this on Windows XP.

1. Log on to Windows as an administrator.
2. Open the **Control Panel**.
3. Double-click **Administrative Tools**.
4. Double-click **Local Security Policy**.
5. Open **Local Policies** in the left pane.
6. Double-click **User Rights Assignment**.
7. Double-click the **Lock Pages In Memory** policy in the right pane.
8. Click **Add User Or Group**.
9. Type the name of the user, and then click **OK**.
10. Click **OK** on the **Lock Pages In Memory** window.
11. Close all open windows and restart the computer for the setting to take effect.

If you specify the -cw option and the -c option on the command line, the database server attempts the initial cache allocation as follows:

1. The AWE cache is no larger than the cache size specified by the -c option. If the value specified by the -c option is less than 2 MB, AWE isn't used.
2. The AWE cache is no larger than all available physical memory less 128 MB.
3. The AWE cache is no smaller than 2 MB. If this minimum amount of physical memory isn't available, an AWE cache isn't used.

When you specify the -cw option and do not specify the -c option, the database server attempts the initial cache allocation as follows:

1. The AWE cache uses 100% of all available memory except for 128 MB that is left free for the operating system.
2. The AWE cache is no larger than the sum of the sizes of the main database files specified on the command line. Additional dbspaces apart from the main database files aren't included in the calculation. If no files are specified, this value is zero.
3. The AWE cache is no smaller than 2 MB. If this minimum amount of physical memory isn't available, an AWE cache isn't used.

When the server uses an AWE cache, the cache page size will be no smaller than 4 KB and dynamic cache sizing is disabled.

See "Use the cache to improve performance" [*SQL Anywhere Server - SQL Usage*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "-c dbeng12/dbsrv12 server option" on page 159
- "-ca dbeng12/dbsrv12 server option" on page 161
- "-cc dbeng12/dbsrv12 server option" on page 162
- "-ch dbeng12/dbsrv12 server option" on page 163
- "-cl dbeng12/dbsrv12 server option" on page 166
- "-cm dbeng12/dbsrv12 server option" on page 167
- "-cr dbeng12/dbsrv12 server option" on page 169
- "-cs dbeng12/dbsrv12 server option" on page 169
- "-cv dbeng12/dbsrv12 server option" on page 170

Example

The following example starts a database server named **myserver** that starts with a cache size of 12 GB and loads the database *c:\test\mydemo.db*:

```
dbeng12 -n myserver -c 12G -cw c:\test\mydemo.db
```

-dt dbeng12/dbsrv12 server option

Specifies the directory where temporary files are stored.

Syntax

{ **dbeng12** | **dbsrv12** } **-dt** *temp-file-dir* ...

Applies to

All servers and operating systems, except shared memory connections on Unix.

Remarks

SQL Anywhere creates two types of temporary files: database server-related temporary files (created on all platforms), and communications-related temporary files (created only on Unix for both the client and the database server).

You can use the -dt option to specify a directory for database server-related temporary files. If you do not specify this option when starting the database server, SQL Anywhere examines the following environment variables, in the order shown, to determine the directory in which to place the temporary file.

- SATMP
- TMP
- TMPDIR
- TEMP

If none of the environment variables are defined, SQL Anywhere places its temporary file in the current directory on Windows, and in the */tmp* directory on Unix.

Temporary files for communications on Unix are not placed in the directory specified by -dt. Instead, the environment variables are examined, and */tmp* is used if none of the environment variables are defined.

See also

- "Working with database files" on page 4
- "SATMP environment variable" on page 385
- "Place different files on different devices" [*SQL Anywhere Server - SQL Usage*]
- "sa_disk_free_space system procedure" [*SQL Anywhere Server - SQL Reference*]
- "temp_space_limit_check option" on page 602

-ec dbeng12/dbsrv12 server option

Uses transport-layer security or simple encryption to encrypt all native SQL Anywhere packets (DBLib, ODBC, and OLE DB) transmitted to and from all clients. TDS packets aren't encrypted.

Syntax

{ **dbeng12** | **dbsrv12** } **-ec** *encryption-options* ...

encryption-options :

{ **NONE** |
  SIMPLE |
  TLS (TLS_TYPE=*cipher***;**
  [FIPS={ **Y** | **N** }**;]**
  IDENTITY=*server-identity-filename***;**
  IDENTITY_PASSWORD=*password* **)** }**,** ...

Allowed values

- **NONE** accepts connections that aren't encrypted.
- **SIMPLE** accepts connections that are encrypted with simple encryption. This type of encryption is supported on all platforms, and on previous versions of SQL Anywhere. Simple encryption doesn't provide server authentication, strong elliptic-curve or RSA encryption, or other features of transport-layer security.
- **TLS** accepts connections that are encrypted. The TLS parameter accepts the following required arguments:
  - ***cipher*** can be **RSA** or **ECC** for RSA and ECC encryption, respectively. For FIPS-approved RSA encryption, specify **TLS_TYPE=RSA;FIPS=Y**. RSA FIPS uses a separate approved library, but is compatible with clients specifying RSA with SQL Anywhere 9.0.2 or later.

    For a list of supported platforms for FIPS, see http://www.sybase.com/detail?id=1061806.

    The cipher must match the encryption (ECC or RSA) used to create your certificates.

    For information about enforcing the FIPS-approved algorithm, see "-fips dbeng12/dbsrv12 server option" on page 182.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Note
Version 10 and later clients cannot connect to version 9.0.2 or earlier database servers using the ECC algorithm. If you require strong encryption for this configuration, use the RSA algorithm.

- ***server-identity-filename*** is the path and file name of the server identity certificate. If you are using FIPS-approved RSA encryption, you must generate your certificates using the RSA cipher.

  For more information about creating the server certificate, which can be self-signed, or signed by a Certificate Authority or enterprise root certificate, see "Creating digital certificates" on page 1146.

- ***password*** is the password for the server private key. You specify this password when you create the server certificate.

Applies to

NONE and SIMPLE apply to all servers and operating systems.

TLS applies to all servers and operating systems, except Windows Mobile.

For information about FIPS support, see http://www.sybase.com/detail?id=1061806.

Remarks

You can use this option to secure communication packets between client applications and the database server using transport-layer security. See "Transport-layer security" on page 1143.

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

The -ec option instructs the database server to accept only connections that are encrypted using one of the specified types. You must specify at least one of the supported parameters in a comma-separated list. Connections over the TDS protocol, which include Java applications using jConnect, are always accepted and are never encrypted, regardless of the usage of the -ec option. Setting the TDS protocol option to NO disallows these unencrypted TDS connections. See "TDS protocol option" on page 338.

By default, communication packets aren't encrypted, which poses a potential security risk. If you are concerned about the security of network packets, use the -ec option. Encryption affects performance only marginally.

If the database server accepts simple encryption, but does not accept unencrypted connections, then any non-TDS connection attempts using no encryption automatically use simple encryption.

Starting the database server with `-ec SIMPLE` tells the database server to only accept connections using simple encryption. TLS connections (ECC, RSA, and RSA FIPS) fail, and connections requesting no encryption use simple encryption.

Starting the server with `-ec SIMPLE,TLS(TLS_TYPE=ECC)` tells the database server to only accept connections with ECC encryption or simple encryption. Both RSA and RSA FIPS connections fail, and connections requesting no encryption use simple encryption.

If you want the database server to accept encrypted connections over TCP/IP, but also want to be able to connect to the database from the local computer over shared memory, you can specify the -es option with the -ec option when starting the database server. See "-es dbeng12/dbsrv12 server option" on page 180.

The *dbecc12.dll* and *dbrsa12.dll* files contain the ECC and RSA code used for encryption and decryption. The file *dbfips12.dll* contains the code for the FIPS-approved RSA algorithm. When you connect to the database server, if the appropriate file cannot be found, or if an error occurs, a message appears in the database server messages window. The server doesn't start if the specified types of encryption cannot be initiated.

The client's and the server's encryption settings must match or the connection will fail except in the following cases:

- if -ec SIMPLE is specified on the database server, but -ec NONE is not, then connections that do not request encryption can connect and automatically use simple encryption.
- if the database server specifies RSA and the client specifies FIPS, or vice versa, the connection succeeds. In this case, the Encryption connection property returns the value specified by the database server.

See also

- "Starting the database server with transport-layer security" on page 1152
- "Encryption (ENC) connection parameter" on page 287
- "-ek dbeng12/dbsrv12 database option" on page 256
- "-ep dbeng12/dbsrv12 server option" on page 179
- "-es dbeng12/dbsrv12 server option" on page 180
- "DatabaseKey (DBKEY) connection parameter" on page 281
- "FIPS protocol option" on page 325

Example

The following example specifies that connections with no encryption and simple encryption are allowed.

```
dbsrv12 -ec NONE,SIMPLE -x tcpip c:\mydemo.db
```

The following example specifies starts a database server that uses the elliptic-curve server certificate *eccserver.id*.

```
dbsrv12 -ec TLS(TLS_TYPE=ECC;IDENTITY=eccserver.id;IDENTITY_PASSWORD=test) -
x tcpip c:\mydemo.db
```

The following example starts a database server that uses the RSA server certificate *rsaserver.id*.

```
dbsrv12 -ec TLS(TLS_TYPE=RSA;IDENTITY=rsaserver.id;IDENTITY_PASSWORD=test) -
x tcpip c:\mydemo.db
```

The following example starts a database server that uses the FIPS-approved RSA server certificate *rsaserver.id*.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
dbsrv12 -ec
TLS(TLS_TYPE=RSA;FIPS=Y;IDENTITY=rsaserver.id;IDENTITY_PASSWORD=test) -x
tcpip c:\mydemo.db
```

-ep dbeng12/dbsrv12 server option

Prompts the user for the encryption key upon starting a strongly encrypted database.

Syntax

{ **dbeng12** | **dbsrv12** } **-ep** ...

Applies to

All operating systems and database servers.

Remarks

The -ep option instructs the database server to make a window appear for the user to enter the encryption key for database started on the command line that require an encryption key. This server option provides an extra measure of security by never allowing the encryption key to be seen in clear text.

When used with the server, the user is prompted for the encryption key when the following are all true:

- the -ep option is specified
- the server is a Windows personal server, or the server is just starting up
- a key is required to start a database
- the database server is either not a Windows service, or it is a Windows service with the interact with desktop option turned ON
- the server isn't a daemon (Unix)

If you want to secure communication packets between client applications and the database server use the -ec server option and transport-layer security. See “Transport-layer security” on page 1143.

See also

- “Starting the database server with transport-layer security” on page 1152
- “-ec dbeng12/dbsrv12 server option” on page 176
- “-ek dbeng12/dbsrv12 database option” on page 256
- “Encryption (ENC) connection parameter” on page 287
- “DatabaseKey (DBKEY) connection parameter” on page 281

Example

The user is prompted for the encryption key when the *myencrypted.db* database is started:

```
dbsrv12 -ep -x tcpip myencrypted.db
```

-es dbeng12/dbsrv12 server option

Allows unencrypted connections over shared memory.

Syntax

{ **dbeng12** | **dbsrv12** } **-ec** *encryption-options* **-es** …

Applies to

All servers and operating systems, except Windows Mobile.

Remarks

This option is only effective when specified with the -ec option. The -es option instructs the database server to allow unencrypted connections over shared memory. Connections over TCP/IP must use an encryption type specified by the -ec option. This option is useful in situations where you want remote clients to use encrypted connections, but for performance reasons you also want to access the database from the local computer with an unencrypted connection.

See also

- "-ec dbeng12/dbsrv12 server option" on page 176
- "Starting the database server with transport-layer security" on page 1152

Example

The following example specifies that connections with simple encryption and unencrypted connections over shared memory are allowed.

```
dbsrv12 -ec SIMPLE -es -x tcpip c:\mydemo.db
```

-f dbeng12/dbsrv12 server recovery option

Forces the database server to start after the transaction log has been lost.

Syntax

{ **dbeng12** | **dbsrv12** } **-f** …

Applies to

All operating systems and database servers.

Remarks

Caution
This option is for use in recovery situations only.

If there is no transaction log, the database server performs a checkpoint recovery of the database and then shuts down—it doesn't continue to run. You can then restart the database server without the -f option for normal operation.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

If there is a transaction log in the same directory as the database, the database server performs a checkpoint recovery, and a recovery using the transaction log, and then shuts down—it doesn't continue to run. You can then restart the database server without the -f option for normal operation.

Specifying a cache size when starting the server can reduce recovery time.

See also

- "Running in special modes" on page 48
- "Backup and data recovery" on page 887

Example

The following command forces the database server to start and perform a recovery of the database *mydatabase.db*:

```
dbeng12 mydatabase.db -f
```

-fc dbeng12/dbsrv12 server option

Specifies the file name of a DLL (or shared object on Unix) containing the File System Full callback function.

Syntax

{ **dbeng12** | **dbsrv12** } **-fc** *filename* ...

Applies to

All operating systems and database servers.

Remarks

This option can be used to notify users, and possibly take corrective action, when a file system full condition is encountered. If you use the -fc option, the database server attempts to load the specified DLL and resolve the entry point of the callback function during startup. If the SQL Anywhere database server cannot find both the DLL and the entry point, the database server returns an error and shuts down. The DLL is user-supplied and can use the callback to, among other things, invoke a batch file (or shell script on Unix) you have supplied to take diagnostic or corrective action. Alternatively, the callback function itself can perform such an action.

A sample disk full callback function is located in *samples-dir\SQLAnywhere\DiskFull*.

For information about *samples-dir*, see "Samples directory" on page 392.

SQL Anywhere searches for the callback function DLL in the same locations as it searches for other DLLs and files.

For more information about where SQL Anywhere searches for files, see "How SQL Anywhere locates files" on page 393.

When the database server detects a disk full condition, it invokes the callback function (if one has been provided), passing it the following information:

- the name of the dbspace where the condition was triggered
- the operating system-specific error code from the failed operation

The return code from the call to xp_out_of_disk indicates whether the operation that caused the condition should be aborted or retried. If a non-zero value is returned, the operation is aborted, otherwise it is retried. The callback function is invoked repeatedly as long as it returns zero and the file system operation fails.

On Microsoft Windows platforms, if the database server is started with a database server messages window (neither -qi nor -qw have been specified), and a callback DLL is not provided, a window appears when a disk full condition occurs. This window contains the dbspace name and error code, and allows the user to choose whether the operation that caused the disk full condition should be retried or aborted.

On all other operating systems, when -fc isn't specified and a disk full condition is encountered, a fatal error occurs.

You can create system events to track the available disk space of devices holding the database file, the log file, or the temporary file and alert administrators of a disk space shortage.

See "CREATE EVENT statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "Using callback functions" [*SQL Anywhere Server - Programming*]
- "Understanding system events" on page 935
- "max_temp_space option" on page 559
- "temp_space_limit_check option" on page 602

Example

When the database server starts, it attempts to load the *diskfull.dll* DLL.

```
dbeng12 -fc diskfull.dll
```

-fips dbeng12/dbsrv12 server option

Requires that only FIPS-approved algorithms should be used for strong database and communication encryption.

Syntax

{ **dbeng12** | **dbsrv12** } **-fips** …

Applies to

Windows, Unix, and Linux

Remarks

Specifying this option forces all database server encryption to use FIPS-approved algorithms. This option applies to strong database encryption, client/server transport-layer security, and web services transport-

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

layer security. You can still use unencrypted connections and databases when the -fips option is specified, but you cannot use simple encryption.

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

For strong database encryption, the -fips option causes new databases to use the FIPS equivalent of AES and AES256 if they are specified in the ALGORITHM clause of the CREATE DATABASE statement.

When the database server is started with -fips, you can run databases encrypted with AES, AES256, AES_FIPS, or AES256_FIPS strong encryption, but not databases encrypted with simple encryption. Unencrypted databases can also be started on the server when -fips is specified.

The SQL Anywhere security option must be installed on any computer used to run a database encrypted with AES_FIPS or AES256_FIPS.

For SQL Anywhere transport-layer security, the -fips option causes the server to use the FIPS-approved RSA encryption cipher, even if RSA is specified. If ECC is specified, an error occurs because a FIPS-approved elliptic-curve algorithm is not available.

For transport-layer security for web services, the -fips option causes the server to use HTTPS FIPS, even if HTTPS is specified.

When you specify -fips, the ENCRYPT and HASH functions use the FIPS-approved RSA encryption cipher, and password hashing uses the SHA-256 FIPS algorithm rather than the SHA-256 algorithm.

See also

- "Strong encryption" on page 1131
- "Transport-layer security" on page 1143
- "Encrypting SQL Anywhere web services" on page 1156
- "-ec dbeng12/dbsrv12 server option" on page 176
- "ENCRYPT function [String]" [*SQL Anywhere Server - SQL Reference*]
- "HASH function [String]" [*SQL Anywhere Server - SQL Reference*]

-ga dbeng12/dbsrv12 server option

Unloads the database after the last non-HTTP client connection disconnects.

Syntax

{ **dbeng12** | **dbsrv12** } **-ga** ...

Applies to

All operating systems.

Remarks

Specifying this option on the network server causes each database to be unloaded after the last non-HTTP client connection disconnects. In addition to unloading each database after the last non-HTTP connection disconnects, the database server shuts down when the last database is stopped.

If the only connection to a database is an HTTP connection, and the database is configured to stop automatically, when the HTTP connection disconnects, the database is not unloaded. As well, if you specify the -ga option, and the database has an HTTP connection and a command sequence or TDS connection, when the last command sequence or TDS connection disconnects, the database stops automatically, and any HTTP connections are dropped.

See also

- "Rebuilding databases" [*SQL Anywhere Server - SQL Usage*]
- "AutoStop (ASTOP) connection parameter" on page 269

-gb dbeng12/dbsrv12 server option

Sets the server process priority class.

Windows syntax

{ **dbeng12** | **dbsrv12** } **-gb** { **idle** | **normal** | **high** | **maximum** } ...

Unix/Mac OS X syntax

{ **dbeng12** | **dbsrv12** } **-gb** *level* ...

Allowed values

- **Unix** On Unix, the *level* is an integer from -20 to 19. The default value on Unix is the same as the nice value of the parent process. Lower *level* values represent a more favorable scheduling priority. All restrictions placed on setting a nice value apply to the -gb option. For example, on most Unix platforms, only the root user can lower the priority level of a process (for example, changing it from 0 to -1).

- **Windows** On Windows, normal and high are the commonly-used settings. The value idle is provided for completeness, and maximum may interfere with the running of your computer.

Applies to

Windows, Unix, Mac OS X

-gc dbeng12/dbsrv12 server option

Sets the maximum interval between checkpoints.

Syntax

{ **dbeng12** | **dbsrv12** } **-gc** *minutes* ...

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Default

60 minutes

Allowed values

- **minutes** The default value is the setting of the checkpoint_time database option, which defaults to 60 minutes. If a value of 0 is entered, the default value of 60 minutes is used.

Applies to

All operating systems and database servers.

Remarks

Use this option to set the maximum length of time, in minutes, that the database server runs without doing a checkpoint on each database.

Checkpoints generally occur more frequently than the specified time.

See "How the database server decides when to checkpoint" on page 924.

See also

- "checkpoint_time option" on page 519
- "Understanding the checkpoint log" on page 25
- "How the database server decides when to checkpoint" on page 924

-gd dbeng12/dbsrv12 server option

Sets the permissions required to start or stop a database.

Syntax

{ **dbeng12** | **dbsrv12** } **-gd** { **DBA** | **all** | **none** } ...

Allowed values

- **DBA** Only users with DBA authority can start or stop databases.
- **all** All users can start or stop databases.
- **none** Starting and stopping databases isn't allowed except when the database server itself is started and stopped.

Applies to

All operating systems and database servers.

Remarks

This option specifies the level of permission required for a user to cause a new database file to be loaded by the database server, or to stop a database on a running database server.

The default setting is all for the personal database server, and DBA for the network database server. Both uppercase and lowercase syntax are allowed.

When this option is set to DBA, the client application must already have a connection to the database server to start or stop a database. Providing a DBA user ID and password on a new connection is not enough.

You can obtain the setting of the -gd option using the StartDBPermission server property:

```
SELECT PROPERTY ( 'StartDBPermission' );
```

The permissions for stopping a database server are specified by the -gk option. See "-gk dbeng12/dbsrv12 server option" on page 187.

See also

- "Understanding permissions" on page 447
- "STOP DATABASE statement" [*SQL Anywhere Server - SQL Reference*]

Example

The following steps illustrates how to use the -gd option for the network database server.

1. Start the network database server:

   ```
   dbeng12 -su mypwd -gd DBA
   ```

2. Connect to the utility database from Interactive SQL:

   ```
   dbisql -c "UID=DBA;PWD=mypwd;DBN=utility_db"
   ```

3. Start a database:

   ```
   START DATABASE 'demo.db';
   ```

4. Connect to the database:

   ```
   CONNECT USING 'DBN=demo;UID=DBA;PWD=sql';
   ```

-ge dbeng12/dbsrv12 server option

Sets the stack size for external functions.

Syntax

{ **dbeng12** | **dbsrv12** } **-ge** *integer* ...

Default

32 KB

Applies to

Windows

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

Sets the stack size for threads running external functions, in bytes.

See also

- "Controlling threading behavior" on page 51

-gf dbeng12/dbsrv12 server option

Disables firing of triggers by the server.

Syntax

{ **dbeng12** | **dbsrv12** } **-gf** ...

Applies to

All operating systems and database servers.

Remarks

The -gf server option instructs the server to disable the firing of triggers.

See also

- "fire_triggers option" on page 538
- "Introduction to triggers" [*SQL Anywhere Server - SQL Usage*]

-gk dbeng12/dbsrv12 server option

Sets the permissions required to stop the network server and personal server using dbstop.

Syntax

{ **dbeng12** | **dbsrv12** } **-gk** { **DBA** | **all** | **none** } ...

Allowed values

- **DBA** Only users with DBA authority can use dbstop to stop the database server. This is the default for the network server.
- **all** No permissions are required to shut down the database server. This is the default for the personal server.
- **none** The database server cannot be stopped using dbstop.

Applies to

All operating systems and database servers.

See also

- "Stop Server utility (dbstop)" on page 851

-gl dbeng12/dbsrv12 server option

Sets the permission required to load data using LOAD TABLE, and to unload data using UNLOAD or UNLOAD TABLE.

Syntax

{ **dbeng12** | **dbsrv12** } **-gl** { **DBA** | **all** | **none** } ...

Allowed values

- **DBA** Only users with DBA authority can load or unload data from the database.
- **all** All users can load or unload data from the database.
- **none** Data cannot be unloaded or loaded.

Default

The default setting is all for personal database servers on non-Unix operating systems, and DBA for the network database server and the Unix personal server.

Applies to

All operating systems and database servers.

Remarks

Using the UNLOAD TABLE or UNLOAD statement places data in files on the database server computer, and the LOAD TABLE statement reads files from the database server computer.

To control access to the file system using these statements, the -gl server option allows you to control the level of database permission that is required to use these statements.

Both uppercase and lowercase syntax are acceptable.

The default settings reflect the fact that, on non-Unix platforms, the personal database server is running on the current computer, so the user already has access to the file system.

See also

- "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "UNLOAD statement" [*SQL Anywhere Server - SQL Reference*]

-gm dbeng12/dbsrv12 server option

Limits the number of concurrent connections to the database server.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

{ **dbeng12** | **dbsrv12** } **-gm** *integer* ...

Default

The default value for the personal server is 10. The default value for the network database server is 32766, though this number will be reduced by internal temporary connections utilized by the server during operation.

Applies to

All operating systems and database servers.

Remarks

Defines the connection limit for the server. If this number is greater than the number that is allowed under licensing and memory constraints, it has no effect. Computer resources typically limit the number of connections to a network server to a lower value than the default.

The database server allows one extra DBA connection above the connection limit to allow a user with DBA authority to connect to the database server and drop other connections.

See also

- "MaxConnections server property" on page 651

-gn dbsrv12 server option

Sets the multiprogramming level of the database server.

Syntax

dbeng12 -gn *integer* ...

Default

20 active tasks for both the network database server and the personal database server

3 on Windows Mobile

Applies to

All operating systems and database servers.

Remarks

The -gn server option sets the initial database server multiprogramming level.

By default, the network database server automatically adjusts the server multiprogramming level in response to changes in the server's workload. You can manually override the setting of the multiprogramming after a network database server has started by using the sa_server_option system procedure and the CurrentMultiprogrammingLevel parameter. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].

With personal servers, dynamic adjustment of the multiprogramming level is not supported and you cannot change the server's multiprogramming level after the server has been started.

See also

- “Threading in SQL Anywhere” on page 49
- “Configuring the database server's multiprogramming level” on page 52
- “-gna dbsrv12 server option” on page 190
- “-gnh dbsrv12 server option” on page 190
- “-gnl dbsrv12 server option” on page 192
- “sa_server_option system procedure” [*SQL Anywhere Server - SQL Reference*]

-gna dbsrv12 server option

Controls automatic tuning of the database server multiprogramming level.

Syntax

dbsrv12 -gna { **0** | **1** }

Default

1 (Enabled)

Applies to

All operating systems except Windows Mobile.

Network database server only.

Remarks

This option enables and disables dynamic tuning of the database server's multiprogramming level. You can manually change the multiprogramming level while the database server is running using the sa_server_option system.

By default, automatic tuning of the database server's multiprogramming level is enabled.

See also

- “Threading in SQL Anywhere” on page 49
- “Configuring the database server's multiprogramming level” on page 52
- “AutoMultiProgrammingLevel server property” on page 645
- “max_query_tasks option” on page 556
- “-gn dbsrv12 server option” on page 189
- “-gnh dbsrv12 server option” on page 190
- “-gnl dbsrv12 server option” on page 192
- “sa_server_option system procedure” [*SQL Anywhere Server - SQL Reference*]

-gnh dbsrv12 server option

Sets the maximum multiprogramming level - the maximum number of tasks that the database server can execute concurrently.

Syntax

dbsrv12 -gnh *integer* ...

Default

Four times the -gn database server option value

Applies to

All operating systems except Windows Mobile.

Network database server only.

Remarks

This option sets the maximum multiprogramming level of the database server. It limits the maximum number of tasks (both user and system requests) that the database server can execute concurrently. If the database server receives an additional request while at this limit, the new request must wait until an executing task completes. The number of active tasks that can execute simultaneously depends on the number of database server threads and the number of logical processors in use.

The maximum number of combined unscheduled and active requests is limited by the -gm database server option, which limits the number of connections to the database server.

Setting the -gnh value too high can result in errors because a larger portion of the system's address space is consumed for stack space.

The database server's kernel uses tasks as the scheduling unit. The execution of any user request requires at least one task. However, a request may cause the scheduling of additional tasks on its behalf. One example of this behavior is if the request involves the execution of an external procedure or function (Java, Perl, CLR, and so on) that in turn makes database requests back into the database server.

When intra-query parallelism is involved, each access plan component executed in parallel is a task. These tasks count toward the -gnh limit as though they were separate requests. However, tasks created for intra-query parallelism are not reflected in the database properties that track the number of active and inactive requests.

Caution
A stack of the size specified by -gss is allocated for each database server worker. The maximum number of workers is specified by the -gnh option for network servers and the -gn option for personal servers. If you set a large -gss value, and a large value for the maximum number of workers, then the database server may not be able to start, or the size of the cache can be limited significantly. For example if you specified -gss 16M and -gnh 100 when starting the database server, then 1.6 GB of server address space would be reserved for stacks.

See also

- “Threading in SQL Anywhere” on page 49
- “Configuring the database server's multiprogramming level” on page 52
- “MaxMultiProgrammingLevel server property” on page 651
- “max_query_tasks option” on page 556
- “-gn dbsrv12 server option” on page 189
- “-gna dbsrv12 server option” on page 190
- “-gnl dbsrv12 server option” on page 192
- “-gss dbeng12/dbsrv12 server option” on page 195
- “sa_server_option system procedure” [*SQL Anywhere Server - SQL Reference*]

-gnl dbsrv12 server option

Sets the minimum network server multiprogramming level.

Syntax

dbsrv12 -gnl *integer* ...

Default

The minimum of the value of the -gtc server option and the number of logical CPUs on the computer. See “-gtc dbeng12/dbsrv12 server option” on page 196.

Applies to

All operating systems except Windows Mobile.

Network database server only.

Remarks

Setting the -gnl value ensures that the current multiprogramming level cannot be set lower than this value. In workloads that are intensive followed by a period of idle time, it may be beneficial to increase the -gnl value to improve database server performance when intensive processing re-occurs.

See also

- “Threading in SQL Anywhere” on page 49
- “Configuring the database server's multiprogramming level” on page 52
- “MinMultiProgrammingLevel server property” on page 652
- “max_query_tasks option” on page 556
- “-gn dbsrv12 server option” on page 189
- “-gna dbsrv12 server option” on page 190
- “-gnh dbsrv12 server option” on page 190
- “sa_server_option system procedure” [*SQL Anywhere Server - SQL Reference*]

-gns dbsrv12 server option

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Displays statistics for automatic multiprogramming level adjustments in the database server message log.

Syntax

dbsrv12 -gns ...

Default

Statistics are not displayed

Applies to

All operating systems except Windows Mobile.

Network database server only.

Remarks

Multiprogramming level statistics are recorded in the database server message log only when the -gns option is specified.

You can change the setting for the reporting of multiprogramming level statistics once the database server has started using the sa_server_option system procedure. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].

If you want to know whether multiprogramming level statistics are currently being reported, execute the following query:

```
SELECT PROPERTY ( 'AutoMultiProgrammingLevelStatistics' );
```

See also

- "Configuring the database server's multiprogramming level" on page 52
- "AutoMultiProgrammingLevelStatistics server property" on page 645
- "-gn dbsrv12 server option" on page 189
- "-gna dbsrv12 server option" on page 190
- "-gnh dbsrv12 server option" on page 190
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Logging database server actions" on page 41

-gp dbeng12/dbsrv12 server option

Sets the maximum allowed database page size.

Syntax

{ **dbeng12** | **dbsrv12** } **-gp** { **2048** | **4096** | **8192** | **16384** | **32768** } ...

Default

4096 (if a database server is started with no databases loaded)

Applies to

All operating systems and database servers.

Remarks

Database files with a page size larger than the page size of the server cannot be loaded. This option explicitly sets the page size of the server, in bytes.

By default, the server page size is the same as the largest page size of the databases on the command line.

On all platforms, if you do not use this option and start a server with no databases loaded, the default value is 4096.

See also

- "Table and page sizes" [*SQL Anywhere Server - SQL Usage*]
- "Setting a maximum page size" on page 39

-gr dbeng12/dbsrv12 server option

Sets the maximum length of time (in minutes) for recovery from system failure.

Syntax

{ **dbeng12** | **dbsrv12** } **-gr** *minutes* ...

Default

Setting of the recovery_time database option, which defaults to 2 minutes

Applies to

All operating systems and database servers.

Remarks

When a database server is running with multiple databases, the recovery time that is specified by the first database started is used unless overridden by this option.

The value specified by the -gr option instructs the database server how often to perform a checkpoint. For example, if you set -gr to 5, then the database server tries to perform checkpoints often enough so that recovery takes no longer than 5 minutes. However, if recovery is necessary, it runs to completion, even if it takes longer than the length of time specified by -gr.

The recovery time includes both the estimated recovery time and the estimated checkpoint time for the database.

See also

- "recovery_time option" on page 580
- "How the database server decides when to checkpoint" on page 924

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

-gss dbeng12/dbsrv12 server option

Sets the stack size per worker in the database server.

Syntax

{ **dbeng12** | **dbsrv12** } **-gss** { *integer*[**k** | **m**] } ...

Applies to

All operating systems and servers. For Windows, this option is supported on Windows XP and later.

Remarks

The -gss option allows you to lower the memory usage of the database server in environments with limited memory.

The *size* is the amount of memory to use, in bytes. Use **k** or **m** to specify units of kilobytes or megabytes, respectively.

Caution
A stack of the size specified by -gss is allocated for each database server worker. The maximum number of workers is specified by the -gnh option for network servers and the -gn option for personal servers. If you set a large -gss value, and a large value for the maximum number of workers, then the database server may not be able to start, or the size of the cache can be limited significantly. For example if you specified -gss 16M and -gnh 100 when starting the database server, then 1.6 GB of server address space would be reserved for stacks.

The minimum, maximum, and default stack sizes per worker are as follows:

Platform	Minimum	Default	Maximum
Unix (32-bit)	512 KB	512 KB	4 MB
Unix (64-bit)	1 MB	1 MB	8 MB
Windows (32-bit)	64 KB	1 MB	16 MB
Windows (64-bit)	64 KB	4 MB	256 MB
Windows Mobile	64 KB	96 KB	512 KB

See also

- “Threading in SQL Anywhere” on page 49
- “-gss dbeng12/dbsrv12 server option” on page 195

-gt dbeng12/dbsrv12 server option

Sets the maximum number of physical processors that can be used (up to the licensed maximum). This option is only useful on multiprocessor systems.

Syntax

{ **dbeng12** | **dbsrv12** } **-gt** *num-processors* ...

Allowed values

- **num-processors** This integer can be a value between 1 and the minimum of:
  - the number of physical processors on the computer
  - the maximum number of CPUs that the server is licensed for if CPU-licensing is in effect

If the -gt value specified lies outside this range, the lower or upper limit is imposed. For the personal database server (dbeng12) the server uses a -gt value of 1.

Applies to

Windows (except Windows Mobile), Linux, and Solaris.

Remarks

The personal database server is always limited to a single processor. With per-seat licensing, the network database server uses all CPUs available on the computer (the default). With CPU-based licensing, the network database server uses only the number of processors you are licensed for. The number of CPUs that the network database server can use may also be affected by your SQL Anywhere edition. See "Editions and licensing" [*SQL Anywhere 12 - Introduction*].

When you specify a value for the -gt option, the database server adjusts its affinity mask (if supported on that hardware platform) to restrict the database server to run on only that number of physical processors. If the database server is licensed for *n* processors, the server, by default, runs on all logical processors (hyperthreads and cores) of *n* physical processors. This behavior can be further restricted with the -gtc option.

See also

- "-gnh dbsrv12 server option" on page 190
- "-gtc dbeng12/dbsrv12 server option" on page 196
- "Threading in SQL Anywhere" on page 49

-gtc dbeng12/dbsrv12 server option

Controls the maximum processor concurrency that the database server allows.

Syntax

{ **dbeng12** | **dbsrv12** } **-gtc** *logical-processors-to-use* ...

Applies to

Linux, Solaris, and Windows operating systems executing on Intel-compatible x86 and x64 platforms, excluding Windows Mobile.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

When you start the database server, the number of physical and logical processors detected by the database server appears in the database server messages window.

Physical processors are sometimes referred to as **packages** or **dies**, and are the CPUs of the computer. Additional logical processors exist when the physical processors support hyperthreading or are themselves configured as **multiprocessors** (usually referred to as multi-core processors). The operating system schedules threads on logical processors.

The -gtc option allows you to specify the number of logical processors that can be used by the database server. Its effect is to limit the number of database server threads that are created at server startup. This limits the number of active database server tasks that can execute concurrently at any one time. By default, the number of threads created is 1 + the number of logical processors on all licensed physical processors.

By default, the database server allows concurrent use of all logical processors (cores or hyperthreads) on each licensed physical processor. For example, on a single-CPU system that supports hyperthreading, by default the database server permits two threads to run concurrently on one physical processor. If the -gtc option is specified, and the number of logical processors to be used is less than the total available for the number of physical processors that are licensed, then the database server allocates logical processors based on round-robin assignment. Specifying 1 for the -gtc option implicitly disables intra-query parallelism (parallel processing of individual queries). Intra-query parallelism can also be explicitly limited or disabled outright using the max_query_tasks option. See "max_query_tasks option" on page 556.

See also

- "-gnh dbsrv12 server option" on page 190
- "-gt dbeng12/dbsrv12 server option" on page 195
- "Parallelism during query execution" [*SQL Anywhere Server - SQL Usage*]
- "Threading in SQL Anywhere" on page 49

Example

Consider the following examples for a Windows-based SMP computer. In each case, assume a 4-processor system with 2 cores on each physical processor for a total of 8 logical processors. The physical processors are identified with letters and the logical processors (cores in this case) are identified with numbers. This 4-processor system therefore has processing units A0, A1, B0, B1, C0, C1, D0, and D1.

Scenario	Network database server settings
A single CPU license or -gt 1 specified	• -gt 1 • -gtc 2 • -gnh 20 Threads can execute on A0 and A1.

Scenario	Network database server settings
No licensing restrictions on the CPU with -gtc 5 specified	• -gt 4 • -gtc 5 • -gnh 20 Threads can execute on A0, A1, B0, C0, and D0.
A database server with a 3 CPU license and -gtc 5 specified	• -gt 3 • -gtc 5 • -gnh 20 Threads can execute on A0, A1, B0, B1, and C0.
No licensing restrictions on the CPU with -gtc 1 specified	• -gt 4 • -gtc 1 • -gnh 20 Threads can execute only on A0.

-gu dbeng12/dbsrv12 server option

Sets the permission levels for utility commands.

Syntax

{ **dbeng12** | **dbsrv12** } **-gu** { **all** | **none** | **DBA** | **utility_db** } …

Default

DBA

Applies to

All operating systems and database servers.

Remarks

Sets permission levels for utility commands such as CREATE DATABASE and DROP DATABASE.

The utility_db level restricts the use of these commands to only those users who can connect to the utility database.

The value **all** is deprecated.

See also

- "Specifying the permissions required to execute file administration statements" on page 478

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

-im dbeng12/dbsrv12 server option

Runs the database server in memory, reducing or eliminating writes to disk.

Syntax

{ **dbeng12** | **dbsrv12** } **-im** { **c** | **nw** } ...

Allowed values

- **Checkpoint only (-im c)** When running in checkpoint-only mode, the database server does not use a transaction log, so you cannot recover to the most recent committed transaction. However, because the checkpoint log is enabled, the database can be recovered to the most recent checkpoint. Normally when you run a database without a transaction log, the database server still performs a checkpoint on a commit, which affects performance. However, when you run the database server in checkpoint only mode, the database server does not perform a checkpoint after each commit.

  This mode is useful in applications where increased performance is desirable, and the loss of committed transactions after the most recent checkpoint is acceptable.

  The following restrictions apply when running in checkpoint-only mode:

  1. There is no transaction log.
  2. There is no temporary file.
  3. Checkpoints are allowed both on demand and at the database server's normal checkpoint frequency.
  4. Dirty pages are flushed to disk only on checkpoint.

- **Never write (-im nw)** When running in never write mode, committed transactions are not written to the database file on disk. All changes are lost if the database is shut down or crashes, so database files are always left in their original state. Requests to extend or create new dbspaces are allowed, but the changes are not reflected in the database files. You can create and use new dbspaces, but they are not written to disk. Making a backup in never write mode is not useful because any changes to the system dbspace are not written to the file.

  The following restrictions apply when running in never write mode:

  1. There is no transaction log.
  2. There is no checkpoint log.
  3. There is no temporary file.
  4. Dirty database pages are never flushed to disk.
  5. The original database file is never modified.

Applies to

All operating systems and database servers.

Separately licensed component required
In-memory mode requires a separate license. See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

Remarks

This feature is most useful on systems with a large amount of available memory, typically enough to hold all the database files within the cache.

Because changes are never written to the original database files, if a persistent copy of current database contents is required, you must use the dbunload utility or the UNLOAD TABLE statement. You can also use SQL queries to retrieve the changes, but you must then manually write these changes to the database file.

The performance benefits gained from in-memory mode depend on the application workload and the speed of the I/O subsystem. The largest performance gains are seen in applications that insert or update large amounts of data, and in applications that commit and checkpoint frequently.

Often, performance of the in-memory modes is as good as, or better than the performance of using transactional global temporary tables. The smallest performance improvement may be seen with applications that predominately query the database. In general, when using in-memory mode, the best performance can be achieved by pre-growing the cache to an amount large enough to hold the full expected contents of the database files. This eliminates much of the overhead involved in growing the cache in increments while the application is running.

Caution
Since pages are not flushed from cache in never write mode, it is possible to exhaust the available cache if the amount of data in the database grows too large. When this happens, SQL Anywhere issues an error and stops processing requests. For this reason, never write mode should be used with caution, and always with a cache large enough to hold the expected complete working set of pages that an application may use. Since checkpoints continue to occur in the "checkpoint only" mode, there is a reduced risk of the server running out of available cache as compared to the "never write" mode.

For the LOAD TABLE and some ALTER TABLE statements, the checkpoint log is used to partially reverse the effects of a failure or to recover from an error. In never write mode, a checkpoint log is not created and you cannot partially reverse the effects of some statements if they fail or an error occurs. Incorrect or incomplete data could remain in tables. See "Understanding the checkpoint log" on page 25.

See also

- "Separately licensed components" [*SQL Anywhere 12 - Introduction*]
- "-c dbeng12/dbsrv12 server option" on page 159
- "Use in-memory mode" [*SQL Anywhere Server - SQL Usage*]

-k dbeng12/dbsrv12 server option

Controls the collection of Performance Monitor statistics.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

{ **dbeng12** | **dbsrv12** } **-k** ...

Default

Performance Monitor statistics are collected

Applies to

All operating systems and database servers.

Remarks

If you specify -k when you start the database server, then the database server does not collect Performance Monitor statistics. The -k option does not affect the collection of column statistics used by the query optimizer.

This option should only be used in situations where the database server is running on a multi-processor computer where it can be shown by testing to improve performance. For most workloads, the benefit will be negligible, so use of this option is not recommended. When you disable the performance counters, this information is not available for analyzing performance problems.

You can also change the setting for the collection of Performance Monitor statistics using the sa_server_option system procedure. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "CollectStatistics server property" on page 646
- "-ks dbeng12/dbsrv12 server option" on page 204
- "-ksc dbeng12/dbsrv12 server option" on page 204
- "-ksd dbeng12/dbsrv12 server option" on page 205
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Monitoring statistics using Sybase Central Performance Monitor" [*SQL Anywhere Server - SQL Usage*]

-kl dbeng12/dbsrv12 server option

Specifies the file name of the Kerberos GSS-API library (or shared object on Unix) and enables Kerberos authenticated connections to the database server.

Syntax

{ **dbeng12** | **dbsrv12** } **-kl** *GSS-API-library-file* ...

Applies to

All operating systems except Windows Mobile.

Remarks

This option specifies the location and name of the Kerberos GSS-API. This option is only required if the Kerberos client uses a different file name for the Kerberos GSS-API library than the default, or if there are multiple GSS-API libraries installed on the computer running the database server. A Kerberos client must already be installed and configured, and SSPI cannot be used by the database server.

Specifying this option enables Kerberos authentication to the database server.

See also

- "-kr dbeng12/dbsrv12 server option" on page 202
- "-krb dbeng12/dbsrv12 server option" on page 203
- "Kerberos (KRB) connection parameter" on page 294
- "Kerberos authentication" on page 116
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]

Example

The following command starts a database server that uses the *libgssapi_krb5.so* shared object for Kerberos authentication.

```
dbsrv12 -kl libgssapi_krb5.so -n my_server_princ /opt/myapp/kerberos.db
```

-kr dbeng12/dbsrv12 server option

Specifies the realm of the Kerberos server principal and enable Kerberos authenticated connections to the database server.

Syntax

{ **dbeng12** | **dbsrv12** } **-kr** *server-realm* ...

Applies to

All operating systems except Windows Mobile.

Remarks

This option specifies the realm of the Kerberos server principal. Normally, the principal used by the database server for Kerberos authentication is *server-name***@***default-realm*, where *default-realm* is the default realm configured for the Kerberos client. Use this option if you want the server principal to use a different realm than the default realm, in which case the server principal used is *server-name***@***server-realm*.

Specifying this option enables Kerberos authentication to the database server.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "-kl dbeng12/dbsrv12 server option" on page 201
- "-krb dbeng12/dbsrv12 server option" on page 203
- "Kerberos (KRB) connection parameter" on page 294
- "Kerberos authentication" on page 116
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]

Example

The following command starts a database server that accepts Kerberos logins and uses the principal my_server_princ@MYREALM for authentication.

```
dbeng12 -kr MYREALM -n my_server_princ C:\kerberos.db
```

-krb dbeng12/dbsrv12 server option

Enables Kerberos-authenticated connections to the database server.

Syntax

{ **dbeng12** | **dbsrv12** } **-krb** ...

Applies to

All operating systems except Windows Mobile.

Remarks

This option enables Kerberos authentication to the database server. You must specify one or more of the -krb, -kl, and -kr options for the database server to be able to authenticate clients using Kerberos.

Before you can use Kerberos authentication, a Kerberos client must already be installed and configured on both the client and database server computers. Additionally, the principal *server-name@REALM* must already exist in the Kerberos KDC, and the keytab for the principal *server-name@REALM* must already have been securely extracted to the keytab file on the database server computer. The database server will not start if the -krb option is specified, but this setup has not been performed.

Note
The database server name cannot contain any of the following characters: /, \, or @, and database server names with multibyte characters cannot be used with Kerberos.

The login_mode database option must be set to allow Kerberos logins, and Kerberos client principals must be mapped to database user IDs using the GRANT KERBEROS LOGIN statement.

See also

- "-kl dbeng12/dbsrv12 server option" on page 201
- "-kr dbeng12/dbsrv12 server option" on page 202
- "Kerberos (KRB) connection parameter" on page 294
- "Kerberos authentication" on page 116
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]

Example

For a Kerberos principal for the database server named my_server_princ@MYREALM, the following command starts a database server named my_server_princ.

```
dbsrv12 -krb -n my_server_princ C:\kerberos.db
```

-ks dbeng12/dbsrv12 server option

Disables the creation of shared memory that the Performance Monitor uses to collect counter values from the database server.

Syntax

{ **dbeng12** | **dbsrv12** } **-ks 0** ...

Applies to

Windows

Remarks

When you specify this option, the Performance Monitor does not show any database server, database, or connection statistics for the current database server.

See also

- "Monitoring statistics using Sybase Central Performance Monitor" [*SQL Anywhere Server - SQL Usage*]
- "-k dbeng12/dbsrv12 server option" on page 200
- "-ksc dbeng12/dbsrv12 server option" on page 204
- "-ksd dbeng12/dbsrv12 server option" on page 205

-ksc dbeng12/dbsrv12 server option

Specifies the maximum number of connections that the Performance Monitor can monitor.

Syntax

{ **dbeng12** | **dbsrv12** } **-ksc** *integer* ...

Default

10

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Applies to

Windows

See also

- "Monitoring statistics using Sybase Central Performance Monitor" [*SQL Anywhere Server - SQL Usage*]
- "-k dbeng12/dbsrv12 server option" on page 200
- "-ks dbeng12/dbsrv12 server option" on page 204
- "-ksd dbeng12/dbsrv12 server option" on page 205

-ksd dbeng12/dbsrv12 server option

Specifies the maximum number of databases that the Performance Monitor can monitor.

Syntax

{ **dbeng12** | **dbsrv12** } **-ksd** *integer* ...

Default

2

Applies to

Windows

See also

- "Monitoring statistics using Sybase Central Performance Monitor" [*SQL Anywhere Server - SQL Usage*]
- "-k dbeng12/dbsrv12 server option" on page 200
- "-ks dbeng12/dbsrv12 server option" on page 204
- "-ksc dbeng12/dbsrv12 server option" on page 204

-m dbeng12/dbsrv12 server option

Truncates the transaction log when a checkpoint is done.

Syntax

{ **dbeng12** | **dbsrv12** } **-m** ...

Applies to

All operating systems and database servers.

Remarks

This option truncates the transaction log when a checkpoint is done, either at shutdown or as a result of a checkpoint scheduled by the server.

Caution
When this option is selected, there is no protection against media failure on the device that contains the database files.

This option provides a way to automatically limit the growth of the transaction log. Checkpoint frequency is still controlled by the checkpoint_time and recovery_time options (which you can also set on the command line).

The -m option is useful for limiting the size of the transaction log in situations where high volume transactions requiring fast response times are being processed, and the contents of the transaction log aren't being relied upon for recovery or replication. The -m option provides an alternative to operating without a transaction log at all, in which case a checkpoint would be required following each COMMIT and performance would suffer as a result. When the -m option is specified, there is no protection against media failure on the device that contains the database files. Other alternatives for managing the transaction log (for example, using the BACKUP statement and events) should be considered before using the -m option.

To avoid database file fragmentation, it is recommended that where this option is used, the transaction log be placed on a separate device or partition from the database itself.

When this option is used, no operations can proceed while a checkpoint is in progress.

Caution
Do not use the -m option with databases that are being replicated or synchronized. SQL Remote and MobiLink rely on transaction log information.

See also

- "-m dbeng12/dbsrv12 database option" on page 256
- "The transaction log" on page 21
- "Understanding the checkpoint log" on page 25
- "Transaction Log utility (dblog)" on page 862
- "checkpoint_time option" on page 519
- "recovery_time option" on page 580

-n dbeng12/dbsrv12 server option

Sets the name of the database server.

Syntax

{ **dbeng12** | **dbsrv12** } **-n** *server-name database-filename* ...

Default

The name of the first database file (with the path and extension removed) that is started on the database server.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Applies to

All operating systems and database servers.

Remarks

When a database server starts, it attempts to become the default database server on that computer. The first database server to start when there is no default server becomes the default database server. Shared memory connection attempts on that computer that do not explicitly specify a database server name connect to the default server.

Note
It is recommended that you use the -xd option for database servers being used by deployed applications, and that all clients explicitly specify the name of the database server to which they should connect by using the ServerName (Server) connection parameter. This ensures that the database connects to the correct database server when a computer is running multiple SQL Anywhere database servers.

There is no character set conversion performed on the server name. If the client character set and the database server character set are different, using extended characters in the server name can cause the server to not be found. If your clients and servers are running on different operating systems or locales, you should use 7-bit ASCII characters in the server name. See "Connection strings and character sets" on page 410.

Database server names must be valid identifiers. Long database server names are truncated to different lengths depending on the protocol. Database server names cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons, forward slashes (/) or backslashes (\)
- be longer than 250 bytes

Note
On Windows and Unix, version 9.0.2 and earlier clients cannot connect to version 10.0.0 and later database servers with names longer than the following lengths:

- 40 bytes for Windows shared memory
- 31 bytes for Unix shared memory
- 40 bytes for TCP/IP

The server name specifies the name to be used in the ServerName (Server) connection parameter of client application connection strings or profiles. With shared memory, unless -xd is specified, a default database server is used if a server name is not specified and there is at least one database server running on the computer.

Running multiple database servers with the same name is not recommended.

There are two -n options
The -n option is positional. If it appears before any database file names, it is a server option and names the server. If it appears after a database file name, it is a database option and names the database.

For example, the following command names the database server SERV and the database DATA:

```
dbsrv12 -n SERV sales.db -n DATA
```

See "-n dbeng12/dbsrv12 database option" on page 257.

See also

- "Identifiers" [*SQL Anywhere Server - SQL Reference*]
- "ServerName (Server) connection parameter" on page 306
- "Naming the database server and the databases" on page 39
- "-xd dbeng12/dbsrv12 server option" on page 239

Example

If the database server is started on the file *samples-dir\demo.db* and no -n option is specified, the name of the server is demo.

-o dbeng12/dbsrv12 server option

Prints all database server messages to the database server message log file.

Syntax

{ **dbeng12** | **dbsrv12** } **-o** *filename* ...

Applies to

All operating systems and database servers.

Remarks

Print all database server messages, including informational messages, errors, warnings, and MESSAGE statement output, to the specified file, and to the database server messages window. If you specify the -qi option with -o, all messages appear only in the database server message log file.

It is recommended that you do not end the file name with *.log* because this can create problems for utilities that perform operations using the transaction log.

You can obtain the name of the database server message log file by executing the following command:

```
SELECT PROPERTY ( 'ConsoleLogFile' );
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "ConsoleLogFile server property" on page 647
- "Logging database server actions" on page 41
- "-oe dbeng12/dbsrv12 server option" on page 209
- "-on dbeng12/dbsrv12 server option" on page 209
- "-os dbeng12/dbsrv12 server option" on page 210
- "-ot dbeng12/dbsrv12 server option" on page 211
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "-qi dbeng12/dbsrv12 server option" on page 214

-oe dbeng12/dbsrv12 server option

Specifies a file name to log startup errors, fatal errors, and assertions.

Syntax

{ **dbeng12** | **dbsrv12** } **-oe** *filename* ...

Applies to

All operating systems and database servers.

Remarks

Each line in the output log file is prefixed with the date and time. Startup errors include such errors as:

- Couldn't open/read database file: *database file*
- A database server with that name has already started

Fatal errors and assertions are logged to the Windows Application Event Log (except on Windows Mobile) or the Unix system log regardless of whether -oe is specified.

It is recommended that you do not end the file name with *.log* because this can create problems for utilities that perform operations using the transaction log.

See also

- "-o dbeng12/dbsrv12 server option" on page 208
- "-on dbeng12/dbsrv12 server option" on page 209
- "-os dbeng12/dbsrv12 server option" on page 210
- "-ot dbeng12/dbsrv12 server option" on page 211
- "-qi dbeng12/dbsrv12 server option" on page 214

-on dbeng12/dbsrv12 server option

Specifies a maximum size for the database server message log, after which the file is renamed with the extension *.old* and a new file is started.

Syntax

{ **dbeng12** | **dbsrv12** } **-on** { *size*[**k** | **m** | **g**] } ...

Applies to

All operating systems and database servers.

Remarks

The *size* is the maximum file size for the database server message log, in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes respectively. The minimum size limit is 10 KB. By default, there is no maximum size limit.

When the database server message log reaches the specified size, the database server renames the file with the extension *.old*, and starts a new file with the original name.

Note
If the *.old* database server message log file already exists, it is overwritten. To avoid losing old database server message log files, use the -os option instead.

This option cannot be used with the -os option.

It is recommended that you do not end the database server message log file name with *.log* because this can create problems for utilities that perform operations using the transaction log.

See also

- "ConsoleLogMaxSize server property" on page 647
- "Logging database server actions" on page 41
- "-o dbeng12/dbsrv12 server option" on page 208
- "-oe dbeng12/dbsrv12 server option" on page 209
- "-os dbeng12/dbsrv12 server option" on page 210
- "-ot dbeng12/dbsrv12 server option" on page 211
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

-os dbeng12/dbsrv12 server option

Specifies a maximum size for the database server message log file, at which point the file is renamed.

Syntax

{ **dbeng12** | **dbsrv12** } **-os** { *size*[**k** | **m** | **g**] } ...

Applies to

All operating systems and database servers.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

The *size* is the maximum file size for logging database server messages, in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes respectively. The minimum size limit is 10 KB. By default, there is no maximum size limit.

Before the database server logs output messages to the database server message log file, it checks the current file size. If the log message will make the file size exceed the specified size, the database server renames the database server message log file to *yymmddxx.slg*, where *yymmdd* represents the year, month, and day the file was created, and *xx* is a number that starts at 00 and continues incrementing.

This option allows you to identify old database server message log files that can be deleted to free up disk space.

This option cannot be used with the -on option.

It is recommended that you do not end the database server message log file name with *.log* because this can create problems for utilities that perform operations using the transaction log.

See also

- "Logging database server actions" on page 41
- "-o dbeng12/dbsrv12 server option" on page 208
- "-oe dbeng12/dbsrv12 server option" on page 209
- "-on dbeng12/dbsrv12 server option" on page 209
- "-ot dbeng12/dbsrv12 server option" on page 211

-ot dbeng12/dbsrv12 server option

Truncates the database server message log file and appends output messages to it.

Syntax

{ **dbeng12** | **dbsrv12** } **-ot** *logfile* ...

Applies to

All operating systems and database servers.

Remarks

The functionality is the same as the -o option except the database server message log file is truncated before any messages are written to it. You can obtain the name of the database server message log file using the following command:

```
SELECT PROPERTY ( 'ConsoleLogFile' );
```

It is recommended that you do not end the database server message log file name with *.log* because this can create problems for utilities that perform operations using the transaction log.

See also

- "Logging database server actions" on page 41
- "-o dbeng12/dbsrv12 server option" on page 208
- "-oe dbeng12/dbsrv12 server option" on page 209
- "-on dbeng12/dbsrv12 server option" on page 209
- "-os dbeng12/dbsrv12 server option" on page 210

-p dbeng12/dbsrv12 server option

Sets the maximum size of communication packets.

Syntax

{ **dbeng12** | **dbsrv12** } **-p** *integer* ...

Default

7300 bytes (all operating systems except Windows Mobile)

1460 bytes (Windows Mobile)

Applies to

All operating systems and database servers.

Remarks

The minimum value is 500 bytes and the maximum value is 16000.

You can change the communication buffer size for a connection by setting the CommBufferSize (CBSIZE) connection parameter.

See also

- "-pc dbsrv12 server option" on page 212
- "-pt dbsrv12 server option" on page 213
- "CommBufferSize (CBSIZE) connection parameter" on page 271

-pc dbsrv12 server option

Compresses all connections except for same-computer connections.

Syntax

dbsrv12 -pc ...

Applies to

All operating systems and network servers, except web servers.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

The packets sent between a SQL Anywhere client and server can be compressed using the -pc option. Compressing a connection may improve performance under some circumstances. Large data transfers with highly compressible data tend to get the best compression rates. This option can be overridden for a particular client by specifying COMPRESS=NO in the client's connection parameters.

By default, connections are not compressed. Specifying the -pc option compresses all connections except same-computer connections, web services connections, and TDS connections. TDS connections (including jConnect) do not support SQL Anywhere communication compression.

Same-computer connections over any communication link are not compressed, even if the -pc option or COMPRESS=YES connection parameter is used.

See also

- "-p dbeng12/dbsrv12 server option" on page 212
- "-pt dbsrv12 server option" on page 213
- "Adjusting communication compression settings to improve performance" on page 84
- "Compress (COMP) connection parameter" on page 276
- "Use compression carefully" [*SQL Anywhere Server - SQL Usage*]

-pt dbsrv12 server option

Increases or decreases the size limit at which packets are compressed.

Syntax

dbsrv12 -pt *size* ...

Default

120 bytes

Applies to

All operating systems and network servers.

Remarks

This parameter takes an integer value representing the minimum byte-size of packets to be compressed. Values less than 80 are not recommended.

Under some circumstances, changing the compression threshold can help performance of a compressed connection by allowing you to compress packets only when compression will increase the speed at which the packets are transferred. The default setting should be appropriate for most cases.

If both client and server specify different compression threshold settings, the client setting applies.

See also

- "-p dbeng12/dbsrv12 server option" on page 212
- "-pc dbsrv12 server option" on page 212
- "Adjusting communication compression settings to improve performance" on page 84
- "CompressionThreshold (COMPTH) connection parameter" on page 277
- "Use compression carefully" [*SQL Anywhere Server - SQL Usage*]

-qi dbeng12/dbsrv12 server option

Controls whether database server system tray icon and database server messages window appear.

Syntax

{ **dbeng12** | **dbsrv12** } **-qi** ...

Applies to

Windows

Remarks

This option leaves no visual indication that the server is running, other than possible startup error windows. You can use either (or both) the -o or -oe log files to diagnose errors.

See also

- "-qn dbeng12/dbsrv12 server option" on page 214
- "-qp dbeng12/dbsrv12 server option" on page 215
- "-qs dbeng12/dbsrv12 server option" on page 216
- "-qw dbeng12/dbsrv12 server option" on page 216
- "-o dbeng12/dbsrv12 server option" on page 208
- "-oe dbeng12/dbsrv12 server option" on page 209

-qn dbeng12/dbsrv12 server option

Specifies that the database server messages window is not minimized on startup.

Syntax

{ **dbeng12** | **dbsrv12** } **-qn** ...

Applies to

Windows

Linux (if X window server is used)

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

By default, the database server messages window automatically minimizes once database server startup completes. When this option is specified, the database server messages window does not minimize after the database server starts.

The database server messages window may appear in the background if an application automatically starts the database server when the application is not active and -qn is specified.

On Linux, you must specify the -ux option (use X window server) with the -qn option.

See also

- "-ux dbeng12/dbsrv12 server option" on page 233
- "-qi dbeng12/dbsrv12 server option" on page 214
- "-qp dbeng12/dbsrv12 server option" on page 215
- "-qs dbeng12/dbsrv12 server option" on page 216
- "-qw dbeng12/dbsrv12 server option" on page 216

Example

The following command starts the database server on Linux or Solaris, displays the database server messages window, and does not minimize the database server messages window once the database server is started:

```
dbeng12 -ux -qn sample.db
```

-qp dbeng12/dbsrv12 server option

Specifies that messages about performance do not appear in the database server messages window.

Syntax

{ **dbeng12** | **dbsrv12** } **-qp** ...

Applies to

All operating systems and database servers.

Remarks

Do not display messages about performance in the database server messages window. Messages that are suppressed include the following:

- No unique index or primary key for table '*table-name*'
- Database file "*mydatabase.db*" consists of *nnn* fragments

See also

- “-qi dbeng12/dbsrv12 server option” on page 214
- “-qn dbeng12/dbsrv12 server option” on page 214
- “-qs dbeng12/dbsrv12 server option” on page 216
- “-qw dbeng12/dbsrv12 server option” on page 216

-qs dbeng12/dbsrv12 server option

Suppresses startup error windows.

Syntax

{ **dbeng12** | **dbsrv12** } **-qs** ...

Applies to

Windows

Remarks

This option suppresses startup error windows. Examples of startup errors include the database server not being able to open or read a database file or a database server not starting because another database server with the specified name is already running.

On Windows platforms, if the server isn't being started automatically, these errors appear in a window and must be cleared before the server stops. These windows do not appear if the -qs option is used.

If there is an error loading the language DLL, no window appears if -qs was specified on the command line and not in @data. This error isn't logged to the -o or -oe logs, but rather to the Windows Application Event Log (except on Windows Mobile).

Usage errors are suppressed if -qs is on the command line, but not in @data expansion.

See also

- “-qi dbeng12/dbsrv12 server option” on page 214
- “-qn dbeng12/dbsrv12 server option” on page 214
- “-qp dbeng12/dbsrv12 server option” on page 215
- “-qw dbeng12/dbsrv12 server option” on page 216
- “-o dbeng12/dbsrv12 server option” on page 208
- “-oe dbeng12/dbsrv12 server option” on page 209

-qw dbeng12/dbsrv12 server option

Specifies that the database server messages window does not appear.

Syntax

{ **dbeng12** | **dbsrv12** } **-qw** ...

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Applies to

All operating systems and database servers.

Remarks

This option suppresses the database server messages window. On Windows platforms, the database server system tray icon is still visible. You can use either (or both) the -o or -oe log files to diagnose errors.

See also

- "-qi dbeng12/dbsrv12 server option" on page 214
- "-qn dbeng12/dbsrv12 server option" on page 214
- "-qp dbeng12/dbsrv12 server option" on page 215
- "-qs dbeng12/dbsrv12 server option" on page 216

-r dbeng12/dbsrv12 server option

Forces all databases that start on the database server to be read-only. No changes to the database(s) are allowed: the database server doesn't modify the database file(s) or transaction log files.

Syntax

{ **dbeng12** | **dbsrv12** } **-r** ...

Applies to

All operating systems and database servers.

Remarks

Opens all database files as read-only with the exception of the temporary file when the option is specified before any database names on the command line. If the -r option is specified after a database name, only that specific database is read-only. You can make changes on temporary tables, but ROLLBACK has no effect, since the transaction and rollback logs are disabled.

A database distributed on a CD-ROM device is an example of a database file that cannot be modified. You can use read-only mode to access this sort of database.

If you attempt to modify the database, for example with an INSERT or DELETE statement, a SQLSTATE_READ_ONLY_DATABASE error is returned.

Databases that require recovery cannot be started in read-only mode. For example, database files created using an online backup cannot be started in read-only mode if there were any open transactions when the backup was started, since these transactions would require recovery when the backup copy is started.

Databases with auditing turned on cannot be started in read-only mode.

If you are checking the validity of a backup copy, you should run the database in read-only mode so that it is not modified in any way. See "Validate a database" on page 930.

See also

- "-r dbeng12/dbsrv12 database option" on page 258
- "auditing option" on page 514
- "Deploying databases on read-only media" [*SQL Anywhere Server - Programming*]
- "Running in special modes" on page 48

Example

The following command starts two databases in read-only mode

```
dbeng12 -r database1.db database2.db
```

The following command starts only the first of two databases in read-only mode.

```
dbeng12 database1.db -r database2.db
```

-s dbeng12/dbsrv12 server option

Sets the user ID for Syslog messages.

Syntax

{ **dbeng12** | **dbsrv12** } **-s** { **none** | **user** | **daemon** | **local***n* } ...

Applies to

Unix, Mac OS X

Remarks

Sets the system user ID used in messages to the Syslog facility. The default is user for database servers that are started in the foreground, and daemon for those that are run in the background (for example, started by dbspawn, started automatically by a client, or started with the -ud database server option).

A value of none prevents any Syslog messages from being logged. The local*n* argument allows you to use a facility identifier to redirect messages to a file. You can specify a number between 0 and 7, inclusive, for *n*. Refer to the Unix Syslog(3) man page for more information.

The following steps illustrate how to redirect messages on Solaris, but you can also do this on Linux, IBM AIX, and Mac OS X. Note that on other platforms, such as HP-UX, the *syslog.conf* file is found in a different location. You can place the */var/adm/sqlanywhere* file in whatever location you want.

To redirect messages to a file using a facility identifier

1. Choose a unique facility identifier that isn't already being used by another application that is running on your system.

   You can do this by looking in the */etc/syslog.conf* file to see of any of the local*n* facilities are referenced.

2. Edit the */etc/syslog.conf* file and add the following line, where local*n* is the facility identifier you chose in step 1:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
local*n*.err;local*n*.info;local*n*.notice  /var/adm/sqlanywhere
```

3. Create the */var/adm/sqlanywhere* file:

   ```
   touch /var/adm/sqlanywhere
   ```

4. Tell the syslogd process that you have modified the *syslog.conf* file by finding the process ID of syslogd:

   ```
   ps -ef | grep syslogd
   ```

   and then executing the following command where *pid* is the process ID of syslogd:

   ```
   kill -HUP pid
   ```

5. Start your SQL Anywhere database server with the following command, where **local*n*** is the facility identifier you chose in step 1:

   ```
   dbeng12 -s local*n* ...
   ```

   Now any messages that the SQL Anywhere database server reports to Syslog are redirected to the */var/adm/sqlanywhere* file.

See also

- "MESSAGE statement" [*SQL Anywhere Server - SQL Reference*]

-sb dbeng12/dbsrv12 server option

Specifies how the server reacts to broadcasts.

Syntax

{ **dbeng12** | **dbsrv12** } **-sb** { **0** | **1** } ...

Applies to

TCP/IP

Remarks

Using -sb 0 causes the database server to not start any UDP broadcast listeners. This forces clients to use a HOST connection parameter or HOST protocol option when connecting to the database server. This also causes the database server to be unlisted when using dblocate.

Using -sb 1 causes the database server to not respond to broadcasts from dblocate, but still starts UDP listeners.

See also

- "BroadcastListener (BLISTENER) protocol option" on page 314

-sf dbeng12/dbsrv12 server option

Enables and disables features for databases running on the current database server.

Syntax

{ **dbeng12** | **dbsrv12** } **-sf** *feature-list* ...

Allowed values

The following *feature-name* values are supported (values enclosed in parentheses are the short forms of feature names that can also be specified):

- **none** Specifies that no features are disabled.
- **all** Disables all features that can be disabled including the following groups.
  - **client** Disables all features that allow access to client-related input/output. This includes access to the client computing environment. This set consists of the following features.
    - **read_client_file** Disables the use of statements that can cause a client file to be read. For example, the READ_CLIENT_FILE function and the LOAD TABLE statement. See "Accessing data on client computers" [*SQL Anywhere Server - SQL Usage*].
    - **write_client_file** Disables the use of all statements that can cause a client file to be written to. For example, the UNLOAD statement and the WRITE_CLIENT_FILE function. See "Accessing data on client computers" [*SQL Anywhere Server - SQL Usage*].
  - **local** Disables all local-related features. This includes access to the server computing environment. This set consists of the **local_call**, **local_db**, **local_io**, and **local_log** feature subsets described below.
    - **local_call** Disables all features that provide the ability to execute code that is not directly part of the server and is not controlled by the server. This set consists of the following features.
      - **cmdshell** Disables the use of the xp_cmdshell procedure. See "xp_cmdshell system procedure" [*SQL Anywhere Server - SQL Reference*].
      - **external_procedure** Disables the use of external stored procedures. This setting does not disable the use of the xp_* system procedures (such as xp_cmdshell, xp_readfile, and so on) that are built into the database server. Separate feature control options are provided for these system procedures. See "SQL Anywhere external call interface" [*SQL Anywhere Server - Programming*].
      - **java** Disables the use of Java-related features, such as Java procedures. See "Creating a Java class for use with SQL Anywhere" [*SQL Anywhere Server - Programming*].
    - **local_db** Disables all features related to database files. This set consists of the following features.
      - **backup** Disables the use of the BACKUP statement, and therefore, the ability to run server-side backups. You can still perform client-side backups using dbbackup. See "BACKUP statement" [*SQL Anywhere Server - SQL Reference*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **restore** Disables the use of the RESTORE DATABASE statement. See "RESTORE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].
- **database** Disables the use of the CREATE DATABASE, ALTER DATABASE, DROP DATABASE, CREATE ENCRYPTED FILE, CREATE DECRYPTED FILE, CREATE ENCRYPTED DATABASE, and CREATE DECRYPTED DATABASE statements.
- **dbspace** Disables the use of the CREATE DBSPACE, ALTER DBSPACE, and DROP DBSPACE statements.

- **local_io** Disables all features that allow direct access to files and their contents. This set consists of the following features.
  - **read_file** Disables the use of statements that can cause a local file to be read. For example, the xp_read_file system procedure, the LOAD TABLE statement, and the use of OPENSTRING(FILE ...). The alternate names load_table and xp_read_file are deprecated.
  - **write_file** Disables the use of all statements that can cause a local file to be written to. For example, the UNLOAD statement and the xp_write_file system procedure. The alternate names unload_table and xp_write_file are deprecated.
  - **delete_file** Disables the use of all statements that can cause a local file to be deleted. For example, it disables the use of the db_delete_file DBLib function, which deletes database files. The db_delete_file function is used by the dbbackup -x and -xo options, so securing db_delete_file causes dbbackup to fail if the -x or -xo options are specified. See "db_delete_file function" [*SQL Anywhere Server - Programming*].
  - **directory** Disables the use of directory class proxy tables. This feature is also disabled when remote_data_access is disabled.

- **local_log** Disables all logging features that result in creating or writing data directly to a file on disk. This set consists of the following features.
  - **request_log** Disables the ability to change the request log file name and also disables the ability to increase the limits of the request log file size or number of files. You can specify the request log file and limits on this file, in the command to start the database server; however, they cannot be changed once the server is started. When request log features are disabled, you can still turn request logging on and off, and reduce the maximum file size and number of request logging files. See "Request logging" [*SQL Anywhere Server - SQL Usage*].
  - **console_log** Disables the ability to change the database server message log file name using the ConsoleLogFile option of the sa_server_option system procedure. It also disables the ability to increase the maximum size of the log file using the ConsoleLogMaxSize option of the sa_server_option system procedure. You can specify a server log file and its size when starting the database server.
  - **webclient_log** Disables the ability to change the web service client log file name using the WebClientLogFile option of the sa_server_option system procedure. You can specify a

web service client log file when starting the database server. See "-zoc dbeng12/dbsrv12 server option" on page 246.

- **remote** Disables all features that allow remote access or communication with remote processes. This set consists of the following features.
  - **remote_data_access** Disables the use of any remote data access services, such as proxy tables.
  - **send_udp** Disables the ability to send UDP packets to a specified address using the sa_send_udp system procedure.
  - **web_service_client** Disables the use of web service client stored procedure calls (that is, stored procedures that issue HTTP requests).

Applies to

All operating systems and database servers.

Remarks

This option allows you to enable and disable features for a database server. These settings affect all databases running on the database server. You can enable all disabled (secured) features for a connection by setting the secure_feature_key option to the key specified by the -sk option. Any connection that sets the secure_feature_key option to the key specified by -sk can also change the set of secured features for a database server using the **SecureFeatures** property of the sa_server_option system procedure.

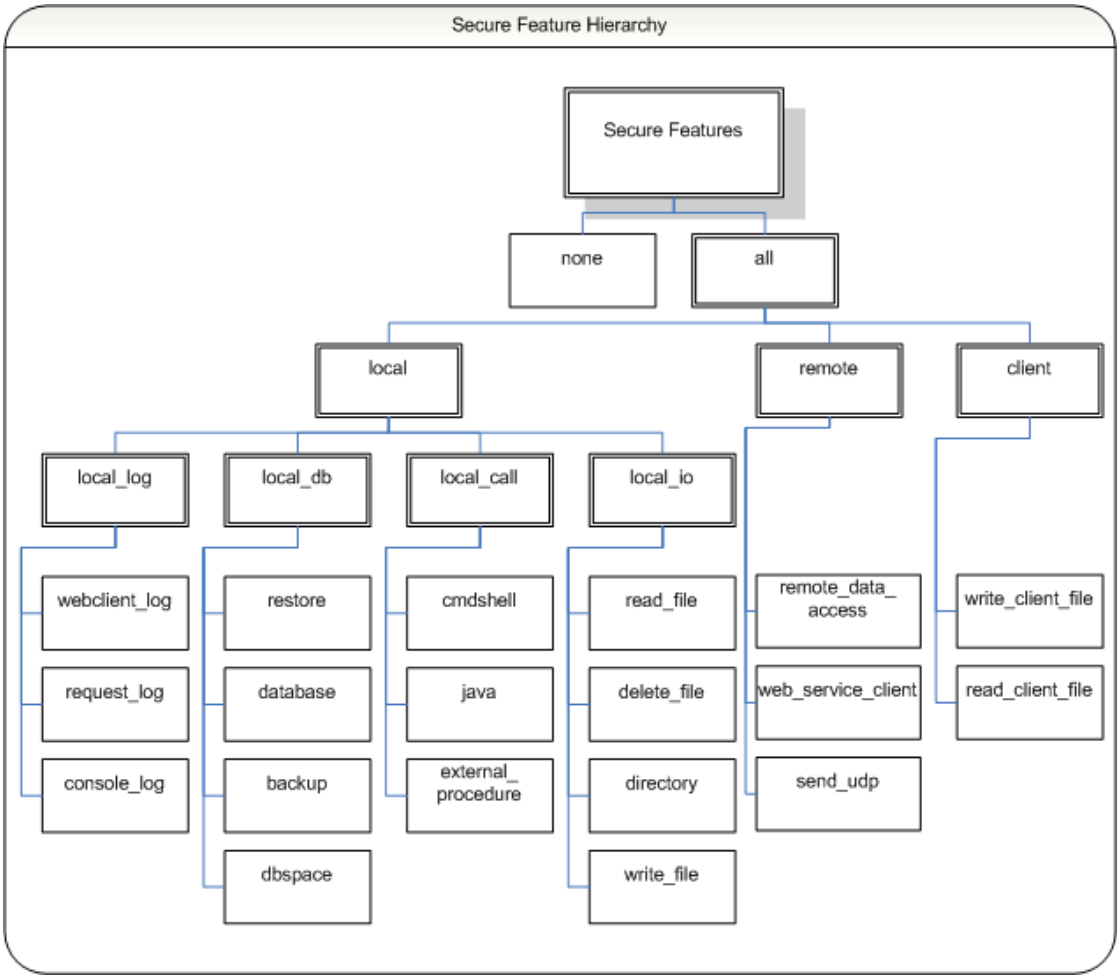
The *feature-list* is a comma-separated list of feature names or feature sets to secure for the database server. Use *feature-name* to indicate that the feature should be disabled, and **-***feature-name* to indicate that the feature should be removed from the disabled features list. For example, the following command indicates that only dbspace features are enabled:

```
dbeng12 -n secure_server -sf all,-dbspace
```

Feature set hierarchy

The following table lists all the feature set keywords and their hierarchy. For example, **local_io** encompasses the **read_file**, **write_file**, **delete_file**, and **directory** features.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0



See also

- "-sk dbeng12/dbsrv12 server option" on page 224
- "secure_feature_key" on page 587
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Specifying secured features" on page 1122

Example

The following command starts a database server named secure_server with access to the request log and with all remote data access features disabled. The key specified by the -sk option can be used later with the secure_feature_key database option to enable these features for a specific connection.

```
dbsrv12 -n secure_server -sf request_log,remote -sk j978kls12
```

If a user connected to a database running on the secure_server database server sets the secure_feature_key option to the value specified by -sk, that connection has access to the request log and remote data access features:

```
SET TEMPORARY OPTION secure_feature_key = 'j978kls12';
```

The following command disables all features, with the exception of local database features:

```
dbeng12 -n secure_server -sf all,-local_db
```

-sk dbeng12/dbsrv12 server option

Specifies a key that can be used to enable features that are disabled for the database server.

Syntax

{ **dbeng12** | **dbsrv12** } **-sk** *key* ...

Applies to

All operating systems and database servers.

Remarks

When you secure features for a database server using the -sf option, you can also include the -sk option, which specifies a key that can be used with the secure_feature_key database option to enable secured features for a connection. That connection can also use the sa_server_option system procedure to modify the features or feature sets that are secured for all databases running on the database server.

If the secure_feature_key option is set to any value other than the one specified by -sk, no error is given, and the features specified by -sf remain secured for the connection.

See also

- "-sf dbeng12/dbsrv12 server option" on page 219
- "secure_feature_key" on page 587
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Specifying secured features" on page 1122

Example

The following command starts a database server named secure_server with access to the backup features disabled. The key specified by the -sk option can be used later to enable these features for a specific connection.

```
dbsrv12 -n secure_server -sf backup -sk j978kls12
```

Setting the secure_feature_key option to the value specified by -sk for a connection to a database running on the secure_server database server allows that connection to perform backups or change the features that are disabled on the secure_server database server:

```
SET TEMPORARY OPTION secure_feature_key = 'j978kls12';
```

The user could then disable the use of all secured features for databases running on secure_server by executing the following command:

```
CALL sa_server_option( 'SecureFeatures', 'all' );
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

-su dbeng12/dbsrv12 server option

Sets the password for the DBA user of the utility database (utility_db), or disable connections to the utility database.

Syntax

{ **dbeng12** | **dbsrv12** } **-su** *password* ...

Applies to

All operating systems and database servers.

Remarks

This option specifies the initial password for the DBA user of the utility database. The password is case sensitive. You can specify **none** for the password to disable all connections to the utility database. To avoid having the utility database password in clear text on the command line, you can use dbfhide to obfuscate a file containing the password, and then reference the obfuscated file on the command line.

If you are using a personal database server and do not specify the -su option, connections to the utility database are allowed with the DBA user ID and any password. If you are using the network database server and do not specify the -su option, connections to the utility database are not allowed unless the *util_db.ini* file exists and the user ID is DBA with a password that matches the password in the *util_db.ini* file. On a network server, if both -su and *util_db.ini* are used, *util_db.ini* is ignored. Note that the *util_db.ini* file is deprecated.

You can execute a CREATE USER DBA IDENTIFIED BY *new-password* statement while connected to utility_db to change the password for the DBA user of the utility database. The REVOKE CONNECT FROM DBA statement can be used to disable connections to the utility_db database.

See also

- "Connecting to the utility database" on page 29
- "File Hiding utility (dbfhide)" on page 794
- "CREATE USER statement" [*SQL Anywhere Server - SQL Reference*]
- "REVOKE statement" [*SQL Anywhere Server - SQL Reference*]

Example

The following command disables all connections to the utility database:

```
dbeng12 -su none c:\inventory.db
```

In the following example, the file named *util_db_pwd.cfg* that contains the utility database password is obfuscated using dbfhide and renamed *util_db_pwd_hide.cfg*:

```
dbfhide util_db_pwd.cfg util_db_pwd_hide.cfg
```

The *util_db_pwd_hide.cfg* file can then be used to specify the utility database password:

```
dbsrv12 -su @util_db_pwd_hide.cfg -n my_server c:\inventory.db
```

-ti dbeng12/dbsrv12 server option

Disconnects inactive connections.

Syntax

{ **dbeng12** | **dbsrv12** } **-ti** *minutes* ...

Default

240 (4 hours)

Applies to

All operating systems and database servers.

Remarks

Disconnect connections that haven't submitted a request for the specified number of *minutes*. The maximum value is 32767. A client computer in the middle of a database transaction holds locks until the transaction is ended or the connection is disconnected. The -ti option is provided to disconnect inactive connections, freeing their locks.

The -ti option is very useful when used in conjunction with dbsrv12 since most connections will be over network links (TCP).

The -ti option is useful with dbeng12 only for local TCP/IP connections. Using -ti has no effect on connections to a local server using shared memory, but you can specify a timeout for the connection that applies to shared memory connections. By default, the personal database server does not start TCP/IP. See "-x dbeng12/dbsrv12 server option" on page 236.

Setting the value to zero disables checking of inactive connections, so that no connections are disconnected. If no connection idle timeout value is set, the idle timeout value is controlled by the setting on the database server. If there is a conflict between timeout values, the connection timeout value supersedes a specified or unspecified database server timeout value.

See also

- "IdleTimeout server property" on page 649
- "-tl dbeng12/dbsrv12 server option" on page 226
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Adjusting timeout values" on page 1005
- "Idle connection parameter" on page 292

-tl dbeng12/dbsrv12 server option

Sets the period at which to send liveness packets.

Syntax

{ **dbeng12** | **dbsrv12** } **-tl** *seconds* ...

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Applies to

All database servers using TCP/IP.

Remarks

A liveness packet is sent periodically across a client/server TCP/IP communications protocol to confirm that a connection is intact. If the server runs for a LivenessTimeout period (default 2 minutes) without detecting a liveness packet on a connection, the communication is severed, and the server drops the connection associated with that client. Unix non-threaded clients and TDS connections do not do liveness checking.

The -tl option on the server sets the LivenessTimeout value for all clients that do not specify a liveness period.

Liveness packets are sent when a connection hasn't sent any packets for between one third and two thirds of the LivenessTimeout value.

When there are more than 200 connections, the server automatically calculates a higher LivenessTimeout value based on the stated LivenessTimeout value, so the server can handle a large number of connections more efficiently. Liveness packets are sent between one third and two thirds of the LivenessTimeout on each idle connection. Large numbers of liveness packets aren't sent at the same time. Liveness packets that take a long time to send could be sent after two thirds of the LivenessTimeout. A warning appears in the database server message log if the liveness sends take a long time. If this warning occurs, consider increasing the LivenessTimeout value.

Although it isn't generally recommended, you can disable liveness by specifying the following:

```
dbsrv12 -tl 0
```

Rather than disabling the LivenessTimeout option, consider increasing the value to 1 hour as follows:

```
dbsrv12 -tl 3600
```

See also

- “LivenessTimeout server property” on page 651
- “-ti dbeng12/dbsrv12 server option” on page 226
- “sa_server_option system procedure” [*SQL Anywhere Server - SQL Reference*]
- “Adjusting timeout values” on page 1005

-tmf dbeng12/dbsrv12 server option

Helps recover from distributed transactions in unusual circumstances.

Syntax

{ **dbeng12** | **dbsrv12** } **-tmf** ...

Applies to

Windows

Remarks

Used during recovery of distributed transactions when the distributed transaction coordinator isn't available. It could also be used if starting a database with distributed transactions in the transaction log, on a platform where the distributed transaction coordinator isn't available.

> **Caution**
> If you use this option, distributed transactions are not recovered properly. It is not intended for routine use.

See also

- "-tmt dbeng12/dbsrv12 server option" on page 228
- "Recovery from distributed transactions" [*SQL Anywhere Server - Programming*]

-tmt dbeng12/dbsrv12 server option

Sets a re-enlistment timeout for participation in distributed transactions.

Syntax

{ **dbeng12** | **dbsrv12** } **-tmt** *milliseconds* ...

Applies to

Windows

Remarks

Used during recovery of distributed transactions. The value specifies how long the database server should wait to be reenlisted. By default there is no timeout (the database server waits indefinitely).

See also

- "-tmf dbeng12/dbsrv12 server option" on page 227
- "Recovery from distributed transactions" [*SQL Anywhere Server - Programming*]

-tq dbeng12/dbsrv12 server option

Shuts down the server at a specified time.

Syntax

{ **dbeng12** | **dbsrv12** } **-tq** { *datetime* | *time* } ...

Applies to

All operating systems and database servers.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

This option is useful for setting up automatic off-line backup procedures. See "Backup and data recovery" on page 887.

The format for the time is in *hh*:*mm* (24 hour clock), and can be preceded by an optional date. If a date is specified, the date and time must be enclosed in double quotes and be in the format *YYYY/MM/DD HH*:*MM*.

See also

- "QuittingTime server property" on page 657
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

-u dbeng12/dbsrv12 server option

Opens files using the operating system disk cache.

Syntax

{ **dbeng12** | **dbsrv12** } **-u** ...

Applies to

Windows, Unix

Remarks

Files are opened using the operating system disk cache in addition to the database cache.

While the operating system disk cache may improve performance, better performance is often obtained by using only the database cache.

If the server is running on a dedicated computer, you shouldn't use the -u option, as the database cache itself is generally more efficient. You may want to use the -u option if the server is running on a computer with several other applications (so that a large database cache may interfere with other applications) and yet IO-intensive tasks are run intermittently on the server (so that a large cache will improve performance).

-ua dbeng12/dbsrv12 server option

Turns off use of asynchronous I/O.

Syntax

{ **dbeng12** | **dbsrv12** } **-ua** ...

Applies to

Linux

Remarks

By default, the database server uses asynchronous I/O on Linux when possible. To use asynchronous I/O, the following conditions must be met:

1. The library *libaio.so* can be loaded at run time.
2. The kernel has asynchronous I/O support.

If you want to turn off the use of asynchronous I/O, specify the -ua option on the database server command line.

-uc dbeng12/dbsrv12 server option

Starts the database server in shell mode. This is the default.

Syntax

{ **dbeng12** | **dbsrv12** } **-uc** ...

Applies to

Unix, Mac OS X

Remarks

Starts the database server in shell mode. You should only specify one of -uc, -ui, -um, or -ux. When you specify -uc, this starts the database server in the same manner as previous releases of the software.

For more information about starting the database server as a daemon, see "-ud dbeng12/dbsrv12 server option" on page 230.

See also

- "-ui dbeng12/dbsrv12 server option" on page 232
- "-um dbeng12/dbsrv12 server option" on page 232
- "-ux dbeng12/dbsrv12 server option" on page 233

-ud dbeng12/dbsrv12 server option

Runs as a daemon.

Syntax

{ **dbeng12** | **dbsrv12** } **-ud** ...

Applies to

Unix, Mac OS X

Remarks

Using this option lets you run the server so that it continues running after the current user session ends.

When you start the daemon directly using the -ud option, the dbeng12 and dbsrv12 commands create the daemon process and return immediately (exiting and allowing the next command to be executed) before the daemon initializes itself or attempts to open any of the databases specified in the command.

One advantage of using dbspawn instead of the -ud option is that the dbspawn process does not shut down until it has confirmed that the daemon has started and is ready to accept requests. If for any reason the daemon fails to start, the exit code for dbspawn is non-zero.

When you start the database server as a daemon, its permissions are controlled by the current user's umask setting. It is recommended that you set the umask value before starting the database server to ensure that the database server has the appropriate permissions.

See also

- "Start Server in Background utility (dbspawn)" on page 849
- "Software component exit codes" [*SQL Anywhere Server - Programming*]
- "Running the database server outside the current session" on page 57
- "Security tips" on page 1116

-uf dbeng12/dbsrv12 server option

Specifies the action to take when a fatal error occurs.

Syntax

{ **dbeng12** | **dbsrv12** } **-uf** *action* ...

Default

default

Allowed values

- **abort** the Unix abort function is called, and a core file is generated.
- **default** the database server always behaves in the same manner as abort, except when a device-full fatal error occurs. In this case, it behaves in the same manner as defunct. This action prevents the system from trying to write a core file on a full device. This is the default behavior.
- **defunct** the database server continues running and does not call abort. Any new connection attempts made to the database server receive the SQL error of the original fatal error.

Applies to

Unix, Mac OS X

See also

- "-oe dbeng12/dbsrv12 server option" on page 209
- "Support utility (dbsupport)" on page 853
- "Error reporting in SQL Anywhere" on page 996
- "Logging database server actions" on page 41

-ui dbeng12/dbsrv12 server option

On Linux this option opens the **Server Startup Options** window, displays the database server messages window, and starts the database server whether or not the X window server starts. On Mac OS X -ui displays database server messages in a new window and starts the database server in shell mode if a usable display isn't available.

Syntax

{ **dbeng12** | **dbsrv12** } **-ui** ...

Applies to

Linux with X window server support, Mac OS X

Remarks

On Linux the -ui option allows you to use the **Server Startup Options** window to specify server options when starting the database server, and to display the database server messages window once the database server has started. On Mac OS X, server messages are redirected to a new window within *DBLauncher.app*.

On Linux, when the -ui option is the only option specified on the server command line, the **Server Startup Options** window appears where you can enter options for starting the database server. On Mac OS X you must use the -ui option with the other options required to start the database server.

The database server attempts to find a usable display when -ui is specified. If it cannot find one, for example because the DISPLAY environment variable isn't set or because X window server isn't running, then the database server starts in shell mode. If you do not want the database server to start when it cannot locate a usable display, specify the -ux option rather than -ui. You should only specify one of -uc, -ui, -um, or -ux.

For information about starting the database server as a daemon, see "-ud dbeng12/dbsrv12 server option" on page 230.

See also

- "-uc dbeng12/dbsrv12 server option" on page 230
- "-um dbeng12/dbsrv12 server option" on page 232
- "-ux dbeng12/dbsrv12 server option" on page 233

-um dbeng12/dbsrv12 server option

Displays database server messages in a new window within *DBLauncher.app*.

Syntax

{ **dbeng12** | **dbsrv12** } **-um** ...

Applies to

Mac OS X

Remarks

The -um option allows you to connect to the *DBLauncher.app* instance, if it is running, and displays messages in a new window within *DBLauncher.app*. The -um option must be used with the other options required to start the database server. Server messages appear in this window instead of in the shell. Closing this window shuts down the database server. If a connection to the *DBLauncher.app* instance cannot be established, the database server does not start.

For the database server to connect to a *DBLauncher.app* instance, both must be running in the same Mac OS X security context. For example, a database server started from an ssh session cannot find a *DBLauncher.app* instance that was started by Launch Services.

For information about starting the database server as a daemon, see "-ud dbeng12/dbsrv12 server option" on page 230.

See also

- "-uc dbeng12/dbsrv12 server option" on page 230
- "-ui dbeng12/dbsrv12 server option" on page 232

-ut dbeng12/dbsrv12 server option

Touches temporary files.

Syntax

{ **dbeng12** | **dbsrv12** } **-ut** *minutes* ...

Default

The default is 30 minutes.

Applies to

Unix, Mac OS X

Remarks

This option causes the server to touch temporary files at specified intervals.

-ux dbeng12/dbsrv12 server option

Opens the **Server Startup Options** window or displays the database server messages window on Linux (use the X window server).

Syntax

{ **dbeng12** | **dbsrv12** } **-ux** ...

Applies to

Linux with X window server support

Remarks

The -ux option allows you to do two things when starting the database server: use the **Server Startup Options** window to specify server options when starting the database server and display the database server messages window once the server has started.

When the -ux option is the only option specified on the server command line, the **Server Startup Options** window appears where you can enter options for starting the database server.

The server must be able to find a usable display when -ux is specified. If it cannot find one, for example because the DISPLAY environment variable isn't set or because X window server isn't running, then the database server fails to start. If you want the database server to start, even if it cannot find a usable display, use the -ui option instead of -ux.

If you specify other server options in addition to -ux, then the database server messages window appears once the database server is started. You should only specify one of -uc, -ui, or -ux.

For more information about starting the database server as a daemon, see "-ud dbeng12/dbsrv12 server option" on page 230.

See also

- "-uc dbeng12/dbsrv12 server option" on page 230
- "-ui dbeng12/dbsrv12 server option" on page 232
- "-qn dbeng12/dbsrv12 server option" on page 214

Example

The following command displays the **Server Startup Options** window where you can enter options for starting the database server:

```
dbeng12 -ux
```

The following command starts the database server and displays the database server messages window:

```
dbeng12 -ux sample.db
```

-v dbeng12/dbsrv12 server option

Displays the software version.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

{ **dbeng12** | **dbsrv12** } **-v** ...

Applies to

All operating systems and database servers.

Remarks

Supplies the database server version in a window, and then stops. You can also obtain the software version by right-clicking the title bar of the database server messages window and choosing **About**.

-vss dbeng12/dbsrv12 server option

Enables and disables the Volume Shadow Copy Service (VSS).

Syntax

{ **dbeng12** | **dbsrv12** } **-vss{ + | - }** ...

Applies to

32-bit Microsoft Windows XP and 32-bit and 64-bit editions of Microsoft Windows 2003 and later operating systems.

Remarks

By default, all SQL Anywhere databases can use the VSS service for backups if the SQL Anywhere VSS writer (*dbvss12.exe*) is running. You can use VSS without the SQL Anywhere VSS writer to back up databases. However, you might need to use the full SQL Anywhere recovery procedures to restore those databases. To prevent a database server from participating in the VSS service, include **-vss-** when starting the database server.

See also

- "Using the SQL Anywhere Volume Shadow Copy Service (VSS)" on page 898
- "Service utility (dbsvc) for Windows" on page 840
- "Recover from media failure on the data" on page 911

Example

The following command starts the *mydatabase.db* database and instructs the database server not to participate in VSS operations even if the (*dbvss12.exe*) writer is running:

```
dbsrv12 -vss- mydatabase.db
```

-wc dbeng12/dbsrv12 server option

Controls whether checksums are enabled on write operations for all databases on this database server if the databases do not have checksums enabled by default.

Syntax

{ **dbeng12** | **dbsrv12** } **-wc**[**+** | **-**] ...

Applies to

All operating systems and servers.

Remarks

The difference between the write checksums (enabled with the -wc option) and global checksums (creating a database with checksums enabled) is that with -wc database pages are checksummed only when they are written out to disk. Pages that are read from disk are only verified if a checksum value was calculated before the pages were written. If a database has checksums enabled, checksums are calculated for all pages when they are written and checksums are verified for all pages when they are read.

If the database server detects that the database is running on Windows Mobile or a removable storage device, such as a network share or USB device, then the database server automatically enables global checksums for all database pages.

By default, databases created with version 10 and 11 of SQL Anywhere do not have global checksums enabled. If you start a database created with SQL Anywhere 11 on a version 12 database server, then by default the database server creates write checksums for pages when they are written to disk (-wc+). Version 12 databases have global checksums enabled by default, so the database server defaults to -wc- for these databases because by default all database pages have checksums. You can use either the -wc option or the START DATABASE statement to change the database server's checksum behavior if you do not want to use the default checksum settings.

You can check whether a database was created with global checksums enabled by executing the following statement:

```
SELECT DB_PROPERTY ( 'Checksum' );
```

You can check whether write checksums are enabled for write I/O operations only by executing the following statement:

```
SELECT DB_PROPERTY ( 'WriteChecksum' );
```

See also

- “Using checksums to detect corruption” on page 928
- “-wc dbeng12/dbsrv12 server option” on page 235
- “START DATABASE statement” [*SQL Anywhere Server - SQL Reference*]

-x dbeng12/dbsrv12 server option

Specifies server side network communications protocols.

Syntax

{ **dbeng12** | **dbsrv12** } **-x** { **all** | **none** | *srv-protocols* } ...

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

srv-protocols:
{ **tcpip** *parmlist* },...
parmlist:
(*parm=value*;...**)**

Allowed values

Regardless of which settings you choose for the -x option, the database server always listens for connection broadcasts using the shared memory protocol.

By default, the personal database server starts only the shared memory protocol, while the network database server starts all available protocols.

You can also specify the following values for the -x option:

- **ALL** Listen for connection attempts by the client using all communications protocols that are supported by the server on this platform, including shared memory. This is the default.
- **NONE** Listen for connection attempts by the client using only the shared memory protocol.
- **TCPIP (TCP)** Listen for connection attempts by the client using the TCP/IP protocol. The TCP/IP protocol is supported by the network server on all operating systems, and by the personal database server for same-computer communications.

  By default, the database server listens for broadcasts on port 2638, and redirects them to the appropriate port. This usually ensures a connection.

  You can override this default and cause the server not to listen on port 2638 by setting the option -sb 0, or by turning off the BroadcastListener option (BroadcastListener=0).

Applies to

All operating systems and database servers.

Remarks

Use the -x option to specify which communications protocols you want to use to listen for client connection broadcasts.

If you do not specify the -x option, the personal database server attempts to listen for client connection broadcasts using shared memory, and the network database server attempts to listen for client broadcasts using shared memory and TCP/IP.

For information about securing shared memory connections on Unix, see "Security tips" on page 1116.

For some protocols, additional parameters may be provided, in the format

```
-x tcpip(PARM1=value1;PARM2=value2;...)
```

For more information about available parameters, see "Network protocol options" on page 311.

For Unix, quotation marks are required if more than one parameter is supplied:

```
-x "tcpip(PARM1=value1;PARM2=value2;...)"
```

See also

- "Examples of starting a database server" on page 34
- "-xa dbsrv12 server option" on page 238
- "-xd dbeng12/dbsrv12 server option" on page 239
- "-xf dbsrv12 server option" on page 239
- "-xp dbsrv12 database option" on page 263
- "-xs dbeng12/dbsrv12 server option" on page 241
- "CommLinks (LINKS) connection parameter" on page 272
- "Using the TCP/IP protocol" on page 78
- "ServerPort (PORT) protocol option" on page 335

Example

Allow shared memory and TCP/IP communications:

```
dbsrv12 -x tcpip
```

-xa dbsrv12 server option

Specifies a comma-separated list of database names and authentication strings for an arbiter server.

Syntax

dbsrv12 -xa auth=*auth-strings*;**DBN=***database-name*

Applies to

All operating systems, network server only.

Remarks

This option is only specified when starting the arbiter server in a database mirroring system.

The authentication string must match the authentication string specified for the primary and mirror servers.

If the lists of authentication strings and database names each contain only one entry, the server will act as the arbiter for only one database mirroring system; otherwise, each list must contain the same number of entries.

See also

- "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]
- "SET MIRROR OPTION statement" [*SQL Anywhere Server - SQL Reference*]
- "Tutorial: Using database mirroring" on page 953
- "Tutorial: Using database mirroring with multiple databases sharing an arbiter server" on page 956
- "DatabaseName (DBN) connection parameter" on page 282
- "-xf dbsrv12 server option" on page 239
- "-xp dbsrv12 database option" on page 263

Example

The following SQL statement defines an arbiter server named arbiter:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
CREATE MIRROR SERVER arbiter
AS ARBITER
connection_string='SERVER=arbiter;HOST=localhost:6870';
```

The following command starts the arbiter database server:

```
dbsrv12
-n arbiter
-su sql
-x tcpip(port=6870)
-xf c:\arbiter\arbiterstate.txt
-xa "AUTH=abc"
```

-xd dbeng12/dbsrv12 server option

Prevents the database server from becoming the default database server.

Syntax

{ **dbeng12** | **dbsrv12** } **-xd** ...

Applies to

All operating systems and database servers.

Remarks

When a database server starts, it attempts to become the default database server on that computer. The first database server to start when there is no default server becomes the default database server. Shared memory connection attempts on that computer that do not explicitly specify a database server name connect to the default server.

Specifying this option prevents the database server from becoming the default database server. If this option is specified, clients that do not specify a database server name cannot find the database server over shared memory. The -xd option also prevents the database server from using the default TCP port. If a TCP port is not specified, the database server uses a port other than port 2638.

See also

- “-n dbeng12/dbsrv12 server option” on page 206
- “StartLine (START) connection parameter” on page 308
- “-x dbeng12/dbsrv12 server option” on page 236
- “ServerName (Server) connection parameter” on page 306
- “Locating a database server” on page 141

-xf dbsrv12 server option

Specifies the location of the file used for maintaining state information about your database mirroring system. This option is only used in the command to start the arbiter server in a database mirroring system.

Syntax

dbsrv12 -xf *state-file* ...

Applies to

All operating systems, network server only.

Remarks

Use the CREATE MIRROR SERVER to define the location of the state information file for the partner servers. See "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*].

The -xf option specifies the location of the file used for maintaining state information about the mirroring system. This option is required for database mirroring.

For more information about the database mirroring state information file, see "State information files" on page 952.

See also

- "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]
- "Tutorial: Using database mirroring" on page 953
- "Tutorial: Using database mirroring with multiple databases sharing an arbiter server" on page 956
- "-xa dbsrv12 server option" on page 238
- "-xp dbsrv12 database option" on page 263

Example

The following command (entered all on one line) starts a database server named myarbiter, that uses the state information file *c:\arbiter\arbiter.state*.

```
dbsrv12 -n myarbiter -su sql
-x "TCPIP(PORT=6870;DOBROAD=no)" -xf "c:\arbiter\arbiter.state"
-xa "AUTH=abc;DBN=mirror_demo"
```

-xm dbeng12/dbsrv12 server option

Specifies how often the database server checks for new IP addresses.

Syntax

{ **dbsrv12** | **dbsrv12** } **-xm** *seconds*

Applies to

All operating systems and database servers.

Remarks

If the computer on which the database server is running is connected to a new network and the -xm option is specified, the change is detected and the database server starts listening for connections on the new network. Also, if the computer is disconnected from a network, the database server stops listening on that network.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The -xm option is disabled by default if the SQL Anywhere application is not running on a portable device. The default setting on portable devices is 120 seconds. To disable the -xm option, specify 0. Use the IsPortableDevice property to check if the SQL Anywhere application is running on a portable device. See "IsPortableDevice server property" on page 650.

Specifying this option does not affect the performance of HTTP or HTTPS listeners.

If you have specified the MyIP (ME) network protocol option, monitoring with the -xm option is disabled.

If a network interface is disconnected, all listeners associated with the network interface are shut down.

Use the sa_server_option system procedure to change the setting of this option without shutting down the database server. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "MyIP (ME) protocol option" on page 333
- "IPAddressMonitorPeriod server property" on page 649

Example

The following command starts a database server that checks for new IP addresses every 60 seconds.

```
dbsrv12 -xm 60;
```

-xs dbeng12/dbsrv12 server option

Specifies server-side web services communications protocols.

Syntax

{ **dbeng12** | **dbsrv12** } **-xs** { *protocol*,... } ...

protocol : {
  NONE
| **HTTP** [(*option*=*value*;...)]
| **HTTPS** [(*option*=*value*;...)]

Allowed values

You can specify any of the following:

- ***option*** For a list of supported *option* values for each protocol, see "Network protocol options" on page 311.
- **HTTP** Listen for web requests by the client using the HTTP protocol. The default port on which to listen is 80.
- **HTTPS** Listen for web requests by the client using the HTTPS protocol. The default port on which to listen is 443. You must specify the server's certificate and password to use HTTPS. The password must be an RSA certificate because HTTPS uses RSA encryption.

The SQL Anywhere HTTP server supports HTTPS connections using SSL version 3.0 and TLS version 1.0.

You can specify **HTTPS**, or **HTTPS** with **FIPS=Y** for FIPS-approved RSA encryption. FIPS-approved HTTPS uses a separate approved library, but is compatible with HTTPS.

Note
The Mozilla Firefox browser can connect when FIPS-approved HTTPS is used. However, the cipher suite used by FIPS-approved HTTPS is not supported by most versions of the Internet Explorer, Opera, or Safari browsers—if you are using FIPS-approved HTTPS, these browsers may not be able to connect.

For information about enforcing the FIPS-approved algorithm, see "-fips dbeng12/dbsrv12 server option" on page 182.

- **NONE** Do not listen for web requests. This is the default.

For more information about available parameters, see "Network protocol options" on page 311.

Applies to

All operating systems and database servers.

Remarks

Use the -xs option to specify which web protocols you want to use to listen for requests.

If you do not specify the -xs option, the database server doesn't attempt to listen for web requests.

If you specify the -xs option with one or more protocols, the server attempts to listen for web requests using the specified protocol(s).

Note
If you want to start multiple web servers at the same time, then you must change the port for one of them since they both have the same default port.

You can use the HTTPS or the FIPS-approved HTTPS protocols for transport-layer security. See "Encrypting SQL Anywhere web services" on page 1156.

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

On Unix, quotation marks are required if more than one parameter is supplied:

```
-xs "HTTP(OPTION1=value1;OPTION2=value2;...)"
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "-sn dbsrv12 database option" on page 260
- "-x dbeng12/dbsrv12 server option" on page 236
- "-xa dbsrv12 server option" on page 238
- "-xf dbsrv12 server option" on page 239
- "-xp dbsrv12 database option" on page 263
- "FIPS protocol option" on page 325
- "Using SQL Anywhere as an HTTP web server" [*SQL Anywhere Server - Programming*]

Example

Listen for HTTP web requests on port 80:

```
dbeng12 web.db -xs HTTP(PORT=80)
```

Listen for web requests using HTTPS:

```
dbeng12 web.db -xs
HTTPS(FIPS=N;PORT=82;IDENTITY=eccserver.id;IDENTITY_PASSWORD=test)
```

-z dbeng12/dbsrv12 server option

Displays diagnostic communication messages, and other messages, for troubleshooting purposes.

Syntax

{ **dbeng12** | **dbsrv12** } **-z** ...

Applies to

All operating systems and database servers.

Remarks

This should only be used when tracking problems. The information appears in the database server messages window.

See also

- "DebuggingInformation server property" on page 647
- "-ze dbeng12/dbsrv12 server option" on page 243
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

-ze dbeng12/dbsrv12 server option

Displays database server environment variables in the database server messages window.

Syntax

{ **dbeng12** | **dbsrv12** } **-ze** ...

Applies to

All operating systems and database servers except Windows Mobile.

Remarks

When you specify the -ze option, environment variables are listed in the database server messages window on startup. You can log the contents of the database server messages window to a file by specifying the -o option when starting the database server.

See also

- "SQL Anywhere environment variables" on page 377
- "-o dbeng12/dbsrv12 server option" on page 208
- "-z dbeng12/dbsrv12 server option" on page 243

Example

The following command starts a database server named myserver, and outputs the environment variables set for the server to the database server messages window and the file *server-log.txt*.

```
dbeng12 -n myserver -ze -o server-log.txt
```

-zl dbeng12/dbsrv12 server option

Turns on capturing of the most recently-prepared SQL statement for each connection to databases on the server.

Syntax

{ **dbeng12** | **dbsrv12** } **-zl** ...

Applies to

All operating systems and database servers.

Remarks

This feature can also be turned on using the RememberLastStatement server setting. You can obtain the most recently-prepared SQL statement for a connection using the LastStatement value of the CONNECTION_PROPERTY function. The sa_conn_activity stored procedure allows you to obtain the most recently-prepared SQL statement for all current connections to databases on the server.

The LastStatement value is set when a statement is prepared, and is cleared when a statement is dropped. Only one statement string is remembered for each connection.

If sa_conn_activity reports a non-empty value for a connection, it is most likely the statement that the connection is currently executing. If the statement had completed, it would likely have been dropped and the property value would have been cleared. If an application prepares multiple statements and retains their statement handles, the LastStatement value does not reflect what a connection is currently doing.

For stored procedure calls, only the outermost procedure call appears, not the statements within the procedure.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Caution
When -zl is specified or when the RememberLastStatement server setting is turned on, any user can call the sa_conn_activity system procedure or obtain the value of the LastStatement connection property to find out the most recently-prepared SQL statement for any other user. This option should be used with caution and turned off when it isn't required.

See also

- "RememberLastStatement server property" on page 657
- "LastStatement connection property" on page 629
- "sa_conn_activity system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

-zn dbeng12/dbsrv12 server option

Specifies the number of request log file copies to retain.

Syntax

{ **dbeng12** | **dbsrv12** } **-zn** *integer*

Applies to

All operating systems and database servers.

Remarks

If request logging is enabled over a long period of time, the request log file can become large. The -zn option allows you to specify the number of request log file copies to retain. It only takes effect if -zs is also specified. The -zs option allows you to create a new log file and rename the original log file when the original log file reaches a specified size. See "-zs dbeng12/dbsrv12 server option" on page 249.

For example, if you redirect request logging information to the file *req.out*, and specify five request log file copies using the -zn option, the server creates files in the following order: *req.out.1*, *req.out.2*, *req.out.3*, *req.out.4*, and *req.out.5*. When these files exist and the active request log fills again, the following happens:

- *req.out.1* is deleted
- the files *req.out.2* to *req.out.5* are renamed *req.out.1* to *req.out.4*
- the copy of the active log is renamed *req.out.5*

Request logging is turned on using the -zr option and redirected to a separate file using the -zo option. You can also set the number of request logs using the sa_server_option system procedure where *nn* specifies the number of request log file copies:

```
CALL sa_server_option('RequestLogNumFiles',nn);
```

See also

- “RequestLogNumFiles server property” on page 657
- “-zo dbeng12/dbsrv12 server option” on page 246
- “-zr dbeng12/dbsrv12 server option” on page 248
- “-zs dbeng12/dbsrv12 server option” on page 249
- “sa_server_option system procedure” [*SQL Anywhere Server - SQL Reference*]
- “Request logging” [*SQL Anywhere Server - SQL Usage*]

Example

In the following example, entered all on one line, request logging information is output to a request log file named *mydatabase.log*, which has a maximum size of 10 KB, and three copies of the request log are kept:

```
dbeng12 "c:\my data\mydatabase.db" -zr all -zn 3
 -zs 10 -zo mydatabase.log
```

-zo dbeng12/dbsrv12 server option

Redirects request logging information to a file separate from the regular log file.

Syntax

{ **dbeng12** | **dbsrv12** } **-zo** *filename* ...

Applies to

All operating systems and database servers.

Remarks

Request logging is turned on using the -zr option. You can direct the output from this file to a different file that is not the regular log file by specifying the -zo option.

This option also prevents request logging from appearing in the database server messages window.

See also

- “RequestLogFile server property” on page 657
- “-zn dbeng12/dbsrv12 server option” on page 245
- “-zr dbeng12/dbsrv12 server option” on page 248
- “-zs dbeng12/dbsrv12 server option” on page 249
- “sa_server_option system procedure” [*SQL Anywhere Server - SQL Reference*]
- “Request logging” [*SQL Anywhere Server - SQL Usage*]

-zoc dbeng12/dbsrv12 server option

Redirects web service client information to a file.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

{ **dbeng12** | **dbsrv12** } **-zoc** *filename* ...

Applies to

All operating systems and database servers.

Remarks

The web service client log file contains HTTP requests and transport data recorded for outbound web service client calls. Logging is enabled automatically when you specify the -zoc server option. You can enable and disable logging to this file using the sa_server_option system procedure:

```
CALL sa_server_option( 'WebClientLogging', 'ON' );
```

See also

- "WebClientLogFile server property" on page 659
- "WebClientLogging server property" on page 659
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Using SQL Anywhere as an HTTP web server" [*SQL Anywhere Server - Programming*]
- "CREATE FUNCTION statement (web clients)" [*SQL Anywhere Server - SQL Reference*]
- "CREATE PROCEDURE statement (web clients)" [*SQL Anywhere Server - SQL Reference*]

Example

The following command starts the database server so that it listens for HTTP web requests on port 80, and logs outbound web service client information to the file *clientinfo.txt*:

```
dbeng12 web.db -xs HTTP(PORT=80) -zoc clientinfo.txt
```

-zp dbeng12/dbsrv12 server option

Turns on capturing of the plan most recently used by the query optimizer.

Syntax

{ **dbeng12** | **dbsrv12** } **-zp** ...

Applies to

All operating systems and database servers.

Remarks

Include this option if you want the database server to store the query execution plan that was used most recently by each connection. This feature can also be turned on using the RememberLastPlan server setting with the sa_server_option system procedure. You can view the text of the most recently-used plan by using the LastPlanText connection property.

See also

- "RememberLastPlan server property" on page 657
- "LastPlanText connection property" on page 629
- "sa_conn_activity system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

-zr dbeng12/dbsrv12 server option

Enables request logging of operations.

Syntax

{ **dbeng12** | **dbsrv12** } **-zr** { **SQL** | **HOSTVARS** | **PLAN** | **PROCEDURES** | **TRIGGERS** | **OTHER** | **BLOCKS** | **REPLACE** | **ALL** | **NONE** } ...

Allowed values

- **SQL** enables logging of the following:
  - START DATABASE statements
  - STOP DATABASE statements
  - STOP SERVER statements
  - Statement preparation and execution
  - EXECUTE IMMEDIATE statement
  - Option settings
  - COMMIT statements
  - ROLLBACK statements
  - PREPARE TO COMMIT operations
  - Connects and disconnects
  - Beginnings of transactions
  - DROP STATEMENT statements
  - Cursor explanations
  - Cursor open, close, and resume
  - Errors
- **PLAN** enables logging of execution plans (short form). Execution plans for procedures are also recorded if logging of procedures (PROCEDURES) is enabled.
- **HOSTVARS** enables logging of host variable values. If you specify HOSTVARS, the information listed for SQL is also logged.
- **PROCEDURES** enables logging of statements executed from within procedures.
- **TRIGGERS** enables logging of statements executed from within triggers.
- **OTHER** enables logging of additional request types not included by SQL, such as FETCH and PREFETCH. However, if you specify OTHER but do not specify SQL, it is the equivalent of specifying SQL+OTHER. Including OTHER can cause the log file to grow rapidly and could negatively impact server performance.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **BLOCKS** enables logging of details showing when a connection is blocked and unblocked on another connection.
- **REPLACE** at the start of logging, the existing request log is replaced with a new (empty) one of the same name. Otherwise, the existing request log is opened and new entries are appended to the end of the file.
- **ALL** logs all supported information. This setting is equivalent to specifying SQL+PLAN +HOSTVARS+PROCEDURES+TRIGGERS+OTHER+BLOCKS. This setting can cause the log file to grow rapidly and could negatively impact server performance.
- **NO or NONE** turns off logging to the request log.

Applies to

All operating systems and database servers.

Remarks

This option should only be used when tracking problems. The information appears in the database server messages window or is sent to the request log. When you specify multiple values, they are separated with a , or a +.

Once the database server is started, you can change the request log settings to log more or less information using the sa_server_option system procedure. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].

You can find the current value of the RequestLogging setting using the following query:

```
SELECT PROPERTY( 'RequestLogging' );
```

See also

- "RequestLogging server property" on page 657
- "-zn dbeng12/dbsrv12 server option" on page 245
- "-zo dbeng12/dbsrv12 server option" on page 246
- "Request logging" [*SQL Anywhere Server - SQL Usage*]

-zs dbeng12/dbsrv12 server option

Limits the size of the request log.

Syntax

{ **dbeng12** | **dbsrv12** } **-zs** { *size*[**k** | **m** | **g**] } ...

Applies to

All operating systems and database servers.

Remarks

Request logging is turned on using the -zr option, and redirected to a separate file using the -zo option. You can limit the size of the file using the -zs option.

The *size* is the maximum file size for the request log, in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes respectively.

If you specify -zs 0, then there is no maximum size for the request logging file, and the file is never renamed. This is the default value.

When the request log file reaches the size specified by either the -zs option or the sa_server_option system procedure, the file is renamed with the extension *.old* appended (replacing an existing file with the same name if one exists). The request log file is then restarted.

See also

- "RequestLogMaxSize server property" on page 657
- "-zn dbeng12/dbsrv12 server option" on page 245
- "-zo dbeng12/dbsrv12 server option" on page 246
- "-zr dbeng12/dbsrv12 server option" on page 248
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Request logging" [*SQL Anywhere Server - SQL Usage*]

Example

The following example shows how the -zs option is used to control log file size. Suppose you start a database server with the following command line:

```
dbeng12 -zr all -zs 10k -zo mydatabase.log
```

A new log file *mydatabase.log* is created. When this file reaches 10 KB in size, any existing *mydatabase.old* files are deleted, *mydatabase.log* is renamed to *mydatabase.old*, and a new *mydatabase.log* file is started. This process is repeated each time the *mydatabase.log* file reaches the specified size (in this case 10 KB).

-zt dbeng12/dbsrv12 server option

Turns on logging of request timing information.

Syntax

{ **dbeng12** | **dbsrv12** } **-zt** ...

Applies to

All operating systems and database servers.

Remarks

Once the database server is started, you can change the status for logging of request timing information using the sa_server_option system procedure. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can find the current value of the RequestTiming setting using the following query:

```
SELECT PROPERTY( 'RequestTiming' );
```

See also

- "RequestTiming server property" on page 658
- "sa_performance_diagnostics system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_performance_statistics system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Request logging" [*SQL Anywhere Server - SQL Usage*]
- "ReqCountActive connection property" on page 637
- "ReqCountBlockContention connection property" on page 637
- "ReqCountBlockIO connection property" on page 637
- "ReqCountBlockLock connection property" on page 637
- "ReqCountUnscheduled connection property" on page 637

Database options

These options are specified after the database file, and apply only to that database.

-a dbeng12/dbsrv12 database option

Applies the named transaction log.

Syntax

{ **dbeng12** | **dbsrv12** } [*server-options*] *database-file* **-a** *log-filename* ...

Applies to

All operating systems and database servers.

Remarks

This option is used to recover from media failure on the database file. When this option is specified, the database server applies the log and then shuts down—it doesn't continue to run. If you need to apply multiple transaction logs, you must know the correct order in which to apply them when using -a. The database server automatically applies multiple transaction logs in the correct order if you use the -ad or -ar option.

The -a database option must be specified after the *database-file*, and applies only to that database.

Specifying a cache size when starting the server can reduce recovery time.

See "Backup and data recovery" on page 887.

See also

- "Recover from media failure on the data" on page 911
- "Recovering a database with multiple transaction logs" on page 907
- "-ad dbeng12/dbsrv12 database option" on page 252
- "-ar dbeng12/dbsrv12 database option" on page 253
- "-as dbeng12/dbsrv12 database option" on page 253

Example

The following example, entered all on one line, applies the log file *demo.log* to a backup copy of the sample database.

```
dbeng12 "c:\backup\demo.db" -a "c:\backup\demo.log"
```

-ad dbeng12/dbsrv12 database option

Specifies the directory containing transaction log files to be applied to the database.

Syntax

{ **dbeng12** | **dbsrv12** } [*server-options*] *database-file* **-ad** *log-directory* ...

Applies to

All operating systems and database servers.

Remarks

When you include the -ad option, the specified directory is scanned for transaction log files associated with the database. Transaction log files with starting log offsets greater than or equal to the start log offset stored in the database file are applied, in log offset order. Once all the transaction log files have been applied, the database is stopped. You must also specify the -as option if you want the database to continue running once the transaction log files have been applied.

The -ad database option must be specified after the *database-file*, and applies only to that database.

See also

- "Recover from media failure on the data" on page 911
- "Recovering a database with multiple transaction logs" on page 907
- "-a dbeng12/dbsrv12 database option" on page 251
- "-ar dbeng12/dbsrv12 database option" on page 253
- "-as dbeng12/dbsrv12 database option" on page 253

Example

The database server applies the log files in the *backup* directory to the *mysample.db* database and then stops the database once the log files have been applied.

```
dbeng12 "c:\mysample.db" -ad "c:\backup"
```

The database server applies the log files in the *backup* directory to the *mysample.db* database and the database continues running once the log files have been applied.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
dbeng12 "c:\mysample.db" -ad "c:\backup" -as
```

-ar dbeng12/dbsrv12 database option

Specifies that any transaction log files located in the same directory as the current transaction log should be applied to the database.

Syntax

{ **dbeng12** | **dbsrv12** } [*server-options*] *database-file* **-ar** ...

Applies to

All operating systems and database servers.

Remarks

When you include the -ar option, the database server looks for transaction log files associated with the database that are located in the same directory as the current transaction log. The transaction log location is obtained from the database. Transaction log files with starting log offsets greater than or equal to the start log offset stored in the database are applied, in log offset order. Once all the transaction log files have been applied, the database is stopped. You must also specify the -as option if you want the database to continue running once the transaction log files have been applied.

The -ar database option must be specified after the *database-file*, and applies only to that database.

See also

- "Recover from media failure on the data" on page 911
- "Recovering a database with multiple transaction logs" on page 907
- "-a dbeng12/dbsrv12 database option" on page 251
- "-ad dbeng12/dbsrv12 database option" on page 252
- "-as dbeng12/dbsrv12 database option" on page 253

Example

The database server applies the transaction log files (whose location is obtained from the database) to the *mysample.db* database. The database continues running after the transaction log files have been applied.

```
dbeng12 "c:\mysample.db" -ar -as
```

-as dbeng12/dbsrv12 database option

Specifies that the database should continue to run after transaction logs have been applied (used in conjunction with -ad or -ar).

Syntax

{ **dbeng12** | **dbsrv12** } [*server-options*] *database-file* { **-ad** *log-dir* | **-ar** } **-as** ...

Applies to

All operating systems and database servers.

Remarks

The -as option must be specified in conjunction with either the -ad or -ar option. When you include -as, the database continues running after the transaction logs are applied to it.

The -as database option must be specified after the *database-file*, and applies only to that database.

See also

- "Recover from media failure on the data" on page 911
- "Recovering a database with multiple transaction logs" on page 907
- "-a dbeng12/dbsrv12 database option" on page 251
- "-ad dbeng12/dbsrv12 database option" on page 252
- "-ar dbeng12/dbsrv12 database option" on page 253

Example

The database server applies the transaction log files to the *mysample.db* database. In this case, because -ar is specified, the database server obtains the location of the transaction logs from the database. The database continues running after the log files have been applied.

```
dbeng12 "c:\mysample.db" -ar -as
```

The database server applies the log files in the *backup* directory to the *mysample.db* database. The database continues running after the log files have been applied.

```
dbeng12 "c:\mysample.db" -ad "c:\backup" -as
```

-ds dbeng12/dbsrv12 database option

Specifies the directory where the dbspaces for the database and the transaction log are located.

Syntax

{ **dbeng12** | **dbsrv12** } **-ds** *dbspace-directory* ...

Applies to

All operating systems and database servers.

Remarks

When a dbspace directory is specified, the database server only searches this directory for dbspaces. The location of the dbspace appears in the database server messages window.

If your backup includes dbspaces with full path names, you can use this option to start the backed up copy of the database on the same computer as the original database while the original database is still running.

The -ds database option must be specified after the *database-file*, and applies only to that database.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

If a transaction log file is not found in the directory specified by this option, then one is created in this location.

Caution
The -ds option should only be used for recovery. If you specify this option and the database has a current, live transaction log that is not located in the directory specified by the -ds option, then a new transaction log is created in the specified location.

See also

- "Using additional dbspaces" on page 16
- "START DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "STOP DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "default_dbspace option" on page 532

Example

The following example starts a database server that looks for dbspaces in the directory *c:\backup\Nov15*:

```
dbeng12 c:\backup\Nov15\my.db -ds c:\backup\Nov15\
```

The following example starts a database server that looks for dbspaces in the current directory:

```
dbeng12 my.db -ds .
```

-dh dbeng12/dbsrv12 database option

Prevents this database from appearing when the Server Enumeration utility (dblocate) is used against this server.

Syntax

{ **dbeng12** | **dbsrv12** } [*server-options*] *database-file* **-dh** ...

Applies to

All platforms.

Remarks

The -dh option makes a database undetectable when the Server Enumeration utility (dblocate) is run against the server. Therefore, when dblocate is used with the -d option, the -dn option, or the -dv option, the database isn't listed.

The -dh database option must be specified after the *database-file*, and applies only to that database.

See also

- "Server Enumeration utility (dblocate)" on page 830

-ek dbeng12/dbsrv12 database option

Specifies the key for a strongly encrypted database.

Syntax

{ **dbeng12** | **dbsrv12** } [*server-options*] *database-file* **-ek** *key* ...

Applies to

All operating systems and servers.

Remarks

You must provide the key value with the -ek option to start an encrypted database. The key is a string, including mixed cases, numbers, letters, and special characters.

If you want to enter the encryption key in a window so it cannot be seen in clear text, use the -ep server option. See "-ep dbeng12/dbsrv12 server option" on page 179.

If you want to secure communication packets between client applications and the database server use the -ec server option and transport-layer security. See "Transport-layer security" on page 1143.

The -ek database option must be specified after the *database-file*, and applies only to that database.

See also

- "-ec dbeng12/dbsrv12 server option" on page 176
- "-ep dbeng12/dbsrv12 server option" on page 179
- "DatabaseKey (DBKEY) connection parameter" on page 281
- "Encrypting and decrypting a database" on page 1130

Example

The following example starts a database and specifies the encryption key on the command line.

```
dbsrv12 -x tcpip mydata.db -ek "Akmm9u70y"
```

-m dbeng12/dbsrv12 database option

Truncates the transaction log when a checkpoint is done.

Syntax

{ **dbeng12** | **dbsrv12** } [*server-options*] *database-file* **-m** ...

Applies to

All operating systems and database servers.

Remarks

Truncates the transaction log when a checkpoint is done, either at shutdown or as a result of a checkpoint scheduled by the server. This option provides a way to limit the growth of the transaction log

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

automatically. Checkpoint frequency is still controlled by the checkpoint_time and recovery_time options (or -gc and -gr database server command line options).

The -m option is useful where high volume transactions requiring fast response times are being processed, and the contents of the transaction log aren't being relied upon for recovery or replication. When this option is selected, there is no protection against media failure on the device that contains the database files.

To avoid database file fragmentation, it is recommended that where this option is used, the transaction log be placed on a separate device or partition from the database itself.

This option is the same as the -m server option, but applies only to the current database or the database identified by the *database-file* variable.

Caution
Do not use the -m option with databases that are being replicated or synchronized. Replication and synchronization, used by SQL Remote and MobiLink, inherently rely on transaction log information.

The -m database option must be specified after the *database-file*, and applies only to that database.

See also

- "-m dbeng12/dbsrv12 server option" on page 205
- "The transaction log" on page 21
- "Transaction Log utility (dblog)" on page 862

Example

The following example starts a database server named silver and loads the database *salesdata.db*. When a checkpoint is done, the transaction log contents are deleted.

```
dbsrv12 -n silver "c:\inventory details\salesdata.db" -m
```

-n dbeng12/dbsrv12 database option

Sets the name of the database.

Syntax

{ **dbeng12** | **dbsrv12** } [*server-options*] *database-file* **-n** *string* ...

Default

Database receives the name of the database file with the path and extension removed

Applies to

All operating systems and database servers.

Remarks

Both database servers and databases can be named. Since a database server can load several databases, the database name is used to distinguish the different databases.

Database names cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons
- be longer than 250 bytes

You can only use the database name utility_db to connect to the SQL Anywhere utility database. See "Using the utility database" on page 28.

The -n database option must be specified after the *database-file*, and applies only to that database.

See also

- "Naming the database server and the databases" on page 39
- "-n dbeng12/dbsrv12 server option" on page 206

Example

If the database that is started is *samples-dir\demo.db* and no -n option is specified, the name of the database is demo.

The following example starts the database server with a cache size of 3 MB, loads the database, and names the database test. Since no database server name has been specified, the server takes its name from the first database, so the server's name is also test.

```
dbsrv12 -c 3MB "c:\mydata.db" -n "test"
```

There are two -n options

The -n option is position dependent. If it appears before a database file name, it is a server option and names the server. If it appears after a database file name, it is a database option and names the database.

For example, the following command names the server SERV and the database DATA:

```
dbsrv12 -n SERV c:\mydata.db -n DATA
```

See "-n dbeng12/dbsrv12 server option" on page 206.

-r dbeng12/dbsrv12 database option

Starts the named database as read-only.

Syntax

{ **dbeng12** | **dbsrv12** } [*server-options*] *database-file* **-r** …

Applies to

All operating systems and database servers.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

Opens all database files (the main database file, dbspaces, transaction log, and transaction log mirrors) as read-only with the exception of the temporary file when the option is specified before any database names on the command line. No changes to the database(s) are allowed: the database server does not modify the database file(s) and transaction log files.

If the -r option is specified after a database file, only that specific database is read-only. You can make changes on temporary tables, but ROLLBACK has no effect, since the transaction and rollback logs are disabled.

A database distributed on a CD-ROM device is an example of a database file that cannot be modified. You can use read-only mode to access this sort of database.

If you attempt to modify the database, for example with an INSERT or DELETE statement, a SQLSTATE_READ_ONLY_DATABASE error is returned.

Databases that require recovery cannot be started in read-only mode. For example, database files created using an online backup cannot be started in read-only mode if there were any open transactions when the backup was started, since these transactions would require recovery when the backup copy is started.

You cannot start a database in read-only mode if auditing is turned on.

See also

- "-r dbeng12/dbsrv12 server option" on page 217
- "auditing option" on page 514

Example

To open two databases in read-only mode

```
dbeng12 -r database1.db database2.db
```

To open only the first of two databases in read-only mode.

```
dbeng12 database1.db -r database2.db
```

-sm dbsrv12 database option (deprecated)

Provides an alternate database server name that can be used to access the read-only mirror database. This option deprecated. Use the CREATE MIRROR SERVER statement instead. See "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

dbsrv12 [*server-options*] *database-file* **-sm** *alternate-server-name*

Applies to

All operating systems, network server only.

Remarks

The *alternate-server-name* is only active when the database server is acting as mirror for the database. By using the -sm and -sn command-line options, an application can always connect to the database on the primary or the mirror server, without knowing which physical server is acting as primary or mirror.

The -sm database option must be specified after the *database-file*, and applies only to that database.

See also

- "Separately licensed components" [*SQL Anywhere 12 - Introduction*]
- "Configuring read-only access to a database running on the mirror server" on page 967
- "-xa dbsrv12 server option" on page 238
- "-xf dbsrv12 server option" on page 239
- "-xp dbsrv12 database option" on page 263
- "START DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "Introduction to database mirroring" on page 945
- "Server Enumeration utility (dblocate)" on page 830
- "ReadOnly database property" on page 673

Example

The following command starts the databases *satest.db* and *sample.db* on a database server named myserver. The -sn option instructs the database server to use mysampleprimary as an alternate server name when connecting to *sample.db*, while the -sm option instructs the database server to use mysamplemirror as an alternate server name to connect to *sample.db*, running on the mirror server.

```
dbsrv12 -n myserver satest.db sample.db -sn mysampleprimary -sm
mysamplemirror
-xp "partner=( Server=server2;LINKS=TCPIP( PORT=2637;TIMEOUT=1 ) );auth=abc;
arbiter=( Server=arbiter;LINKS=TCPIP;( PORT=2639;TIMEOUT=1 ) );mode=sync"
```

You can connect to *sample.db* while it is running on the *primary server* using any of the following connection parameters:

- Server=myserver;DBN=sample
- Server=mysampleprimary
- Server=mysampleprimary;DBN=sample

You cannot connect to *satest.db* using Server=mysampleprimary.

You can connect to *sample.db* while it is running on the *mirror server* using any of the following connection parameters:

- Server=myserver;DBN=sample
- Server=mysamplemirror
- Server=mysamplemirror;DBN=sample

You cannot connect to *satest.db* using Server=mysamplemirror.

-sn dbsrv12 database option

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Provides an alternate server name for a single database running on a database server.

Syntax

dbsrv12 [*server-options*] *database-file* **-sn** *alternate-server-name*

Applies to

All operating systems, network server only.

Remarks

The database server can be configured to listen for more than one server name for a particular database server. Server names other than the real server name are called alternate server names, and are specific to a particular database running on the database server. Clients can specify the alternate server name using the ServerName parameter to connect to that database.

Alternate server names must be unique on the network; otherwise, the database fails to start. If the database is started in the server command and the alternate server name is not unique, the server fails to start. You can also provide an alternate server name using the START DATABASE statement.

Clients that specify an alternate server name can only connect to the database that specified the alternate server name. They cannot connect to, create, stop, and drop other databases on the same database server. If the DBN or DBF connection parameter is specified, it must match the database name or database file, respectively. If the DBN or DBF connection parameter is not specified, then the database acts as the default database for that server.

The Server Enumeration utility (dblocate) detects alternate server names.

Using alternate server names for database mirroring
When using database mirroring, an alternate server name must be specified for client applications to be able to connect to the current primary server without knowing in advance which server is the primary server and which is the mirror server. Both operational servers must use the same name for the alternate server name.

The -sn database option must be specified after the *database-file*, and applies only to that database.

See also

- “Separately licensed components” [*SQL Anywhere 12 - Introduction*]
- “START DATABASE statement” [*SQL Anywhere Server - SQL Reference*]
- “Introduction to database mirroring” on page 945
- “Server Enumeration utility (dblocate)” on page 830
- “AlternateServerName database property” on page 660

Example

The following command starts the databases *satest.db* and *sample.db* on a database server named myserver. The -sn option instructs the database server to use mysample as an alternate server name when connecting to *sample.db*.

```
dbsrv12 -n myserver satest.db sample.db -sn mysample
```

You can connect to *sample.db* using any of the following connection parameters:

- Server=myserver;DBN=sample
- Server=mysample
- Server=mysample;DBN=sample

You cannot connect to *satest.db* using Server=mysample.

-wc dbeng12/dbsrv12 database option

Controls whether checksums are enabled on write operations for the database if it does not have checksums enabled by default.

Syntax

{ **dbeng12** | **dbsrv12** } [*server-options*] *database-file* **-wc**[**+** | **-**]

Applies to

All operating systems and servers.

Remarks

The difference between the write checksums (enabled with the -wc option) and global checksums (creating a database with checksums enabled) is that with -wc database pages are checksummed only when they are written out to disk. Pages that are read from disk are only verified if a checksum value was calculated before the pages were written. If a database has checksums enabled, checksums are calculated for all pages when they are written and checksums are verified for all pages when they are read.

If the database server detects that the database is running on Windows Mobile or a removable storage device, such as a network share or USB device, then the database server automatically enables global checksums for all database pages.

By default, databases created with version 10 and 11 of SQL Anywhere do not have global checksums enabled. If you start a database created with SQL Anywhere 11 on a version 12 database server, then by default the database server creates write checksums for pages when they are written to disk (-wc+). Version 12 databases have global checksums enabled by default, so the database server defaults to -wc- for these databases because by default all database pages have checksums. You can use either the -wc option or the START DATABASE statement to change the database server's checksum behavior if you do not want to use the default checksum settings.

You can check whether a database was created with global checksums enabled by executing the following statement:

```
SELECT DB_PROPERTY ( 'Checksum' );
```

You can check whether write checksums are enabled for write I/O operations only by executing the following statement:

```
SELECT DB_PROPERTY ( 'WriteChecksum' );
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Using checksums to detect corruption" on page 928
- "-wc dbeng12/dbsrv12 server option" on page 235
- "START DATABASE statement" [*SQL Anywhere Server - SQL Reference*]

-xp dbsrv12 database option

Provides information to a server that allows it to connect to its partner and to the arbiter when database mirroring or read-only scale-out is being used. All syntax except **-xp on** is deprecated.

Syntax

dbsrv12 [*server-options*] *database-file* **-xp on**

Syntax (deprecated)

dbsrv12 [*server-options*] *database-file*
-xp { **on** | *mirror-settings* }

mirror-settings :
partner=(*partner-conn*);
auth=*auth-str*;
[;**arbiter=**(*arbiter-conn*)]
[;**mode=**[**sync** | **async** | **page**]
[;**autofailover=**[**YES** | **NO**]]
[;**pagetimeout=***n*]
[;**preferred=**[**YES** | **NO**] ...

Applies to

All operating systems, except Windows Mobile, network server only.

Remarks

When you specify -xp, you must also specify the location of the database mirroring state information file with the -xf option. The -xp database option must be specified after the *database-file*, and applies only to that database.

If the connection parameters specified in the -xp option are invalid, and there are multiple databases running on the server, then the mirrored database fails to start and does not attempt to reconnect. If the mirrored database is the only database running on the database server, then the database server does not start.

on You can only use database mirroring and/or scale-out if you specify -xp option when the database server is started, even if mirroring or scale-out information is stored in the database. When you specify -xp on, you cannot specify other mirroring or scale-out options with the -xp option. The value off is not supported. Database mirroring and scale-out settings are defined in the database using the following statements:

- CREATE MIRROR SERVER
- SET MIRROR OPTION

When you specify **-xp on**, you should also specify the name of the database server in the mirroring system using the -n option, and the password for the utility database using the -su option.

partner-conn (deprecated) Specifies the connection string for the partner server. A user ID and password are not required. It is recommended that you specify a timeout to reduce failover time.

auth-str (deprecated) Specifies the authentication string used by the arbiter.

arbiter-conn (deprecated) Specifies the connection string for the arbiter server. A user ID and password are not required. It is recommended that you specify a timeout to reduce failover time.

mode (deprecated) Specifies the synchronization mode used for database mirroring: synchronous (sync), asynchronous (async), or asyncfullpage (page).

autofailover (deprecated) Specifies whether the mirror server automatically takes over as the primary server when the original primary server goes down. This option does not apply to synchronous mode.

Note
It is recommended that if you are using asynchronous or asyncfullpage mode, that you set the -xp autofailover option to yes. Then, if the primary server goes down, the mirror server automatically takes over as the primary server.

pagetimeout (deprecated) Specifies how often, in seconds, transaction log pages are sent to the mirror server, whether or not they are full. This option applies only when using asyncfullpage mode.

preferred (deprecated) Specifies whether the server is the preferred server in the mirroring system. The preferred server assumes the role of primary server whenever possible. This is equivalent to specifying the preferred option in the CREATE MIRROR SERVER statement. See "Specifying a preferred database server" on page 966.

Caution
In asynchronous and asyncfullpage mode, committed transactions are not guaranteed to be recorded on the mirror server, and it is possible for data to be lost. See "Choosing a database mirroring mode" on page 950.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]
- "SET MIRROR OPTION statement" [*SQL Anywhere Server - SQL Reference*]
- "Setting up a database mirroring system" on page 963
- "SQL Anywhere read-only scale-out" on page 980
- "Tutorial: Setting up a read-only scale-out system" on page 982
- "Tutorial: Using database mirroring" on page 953
- "Tutorial: Using database mirroring with multiple databases sharing an arbiter server" on page 956
- "Separately licensed components" [*SQL Anywhere 12 - Introduction*]
- "Choosing a database mirroring mode" on page 950
- "-sn dbsrv12 database option" on page 260
- "-su dbeng12/dbsrv12 server option" on page 225
- "-xa dbsrv12 server option" on page 238
- "-xf dbsrv12 server option" on page 239
- "-n dbeng12/dbsrv12 server option" on page 206
- "MirrorMode database property" on page 670

Example

The following command starts three database files on a database server that is available to participate in a mirroring system:

```
dbsrv12 -n server1 -x tcpip(PORT=6871) -su sql
c:\server1\one.db -xp on c:\server1\two.db -xp on c:\server1\three.db -xp on
```

Connection parameters

Connection parameters are included in connection strings. They can be entered in the following places:

- In an application's connection string. See "Assembling a list of connection parameters" on page 140 and "Connection parameters passed as connection strings" on page 88.
- In an ODBC data source. See "Creating ODBC data sources" on page 98.
- In the SQL Anywhere **Connect** window. See "Connecting from SQL Anywhere utilities" on page 135.

Notes

- Connection parameters are case insensitive, although their values may not be (for example, file names on Unix).
- Boolean parameters are turned on with YES, Y, ON, TRUE, T, or 1, and are turned off with any of NO, N, OFF, FALSE, F, and 0. The parameters are case insensitive.
- The Usage for each connection parameter describes the circumstances under which the parameter is to be used. Common usage entries include the following:

- **Embedded databases** When SQL Anywhere is used as an embedded database, the connection starts a personal server and loads the database. When the application disconnects from the database, the database is unloaded and the server stops.
- **Running local databases** This refers to the case where a SQL Anywhere personal server is already running, and the database is already loaded on the server.
- **Network servers** When SQL Anywhere is used as a network server, the client application must locate a server already running somewhere on the network and connect to a database.

- You can use the dbping utility to test connection strings. The -c option is used to specify the connection parameters. For example, suppose a personal server with the name demo12 is running the sample database (which can be started with the command `dbeng12 samples-dir\demo.db`). The following string returns a message indicating that the ping was successful if a database server named demo12 is running on the local computer and has a database named demo running:

  ```
  dbping -d -c "Server=demo12;DBN=demo;UID=DBA;PWD=sql"
  ```

  The following command returns the message `Ping database failed - Database server not running` if no database server named other-server is running on the local computer:

  ```
  dbping -d -c "Server=other-server;UID=DBA;PWD=sql"
  ```

  See "Ping utility (dbping)" on page 826.

See also

- "Using connection parameters" on page 86
- "Connection parameter syntax rules" on page 89

AppInfo (APP) connection parameter

Assists administrators in identifying the origin of particular client connections from a database server.

Syntax

{ **AppInfo** | **APP** }=*keyword*=*value*

Usage

Anywhere

Allowed values

Clients can specify their own string that is appended to the generated string. The AppInfo property string is a sequence of semicolon-delimited **key**=*value* pairs. The valid keys are as follows:

- **API** DBLIB, ODBC, OLEDB, ADO.NET, iAnywhereJDBC, PHP, PerlDBD, or DBEXPRESS.
- **APPINFO** If you specified AppInfo in the connection string, the string entered.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **EXE** The name of the client executable (Windows, Linux, and Solaris).
- **HOST** The host name of the client computer.
- **IP** The IP address of the client computer.
- **OS** The operating system name and version number (for example, Windows XP).
- **OSUSER** The operating system user name associated with the client process. If the client process is impersonating another user (or the set ID bit is set on Unix), the impersonated user name is returned. An empty string is returned for version 10.0.1 and earlier clients, and for HTTP and TDS clients.
- **PID** The process ID of the client (Windows and Unix only).
- **THREAD** The thread ID of the client (Windows and Unix only).
- **TIMEZONEADJUSTMENT** The number of minutes that must be added to the Coordinated Universal Time (UTC) to display time local to the connection.
- **VERSION** The version of the client library in use, including major and minor values, and a build number (for example 12.0.0.2413).

Default

Empty string

Remarks

This connection parameter is sent to the database server from embedded SQL, ODBC, OLE DB, or ADO.NET clients and from applications using the SQL Anywhere JDBC driver. It is not available from Sybase Open Client or jConnect applications.

It consists of a generated string that holds information about the client process, such as the IP address of the client computer, the operating system it is running on, and so on. The string is associated in the database server with the connection, and you can retrieve it using the following statement:

```
SELECT CONNECTION_PROPERTY( 'AppInfo' );
```

If you specify a debug log file in your client connection parameters, the AppInfo string is added to the file.

See also

- “Using connection parameters” on page 86
- “request_timeout option” on page 582
- “AppInfo connection property” on page 620

Example

Connect to the sample database from Interactive SQL (the SQL Anywhere JDBC driver is used by default):

```
dbisql -c "UID=DBA;PWD=sql;DBF=samples-dir\demo.db"
```

View the application information:

```
SELECT CONNECTION_PROPERTY( 'AppInfo' );
```

The result is as follows (in a single string):

```
IP=ip-address;
HOST=computer-name;
OSUSER=user-name;
OS='Windows XP Build 2600 Service Pack 2';
EXE='C:\Program Files\SQL Anywhere 12\Bin32\dbisql.exe';P
ID=0xcac;
THREAD=0xca8;VERSION=12.0.0.2413;
API=iAnywhereJDBC;
TIMEZONEADJUSTMENT=-240
```

Connect to the sample database from Interactive SQL, appending your own information to the AppInfo property:

```
dbisql -c "UID=DBA;PWD=sql;DBF=samples-dir\demo.db;APP=Interactive SQL
connection"
```

View the application information:

```
SELECT CONNECTION_PROPERTY( 'AppInfo' );
```

The result is as follows (in a single string):

```
IP=ip-address;
HOST=computer-name;
OSUSER=user-name;
OS=Windows XP Build 2600 Service Pack 2;
EXE=C:\Program Files\SQL Anywhere 12\Bin32\dbisql.exe;
PID=0xcac;
THREAD=0xba8;
VERSION=12.0.0.2413;
API=iAnywhereJDBC;
TIMEZONEADJUSTMENT=-240;
APPINFO='Interactive SQL connection'
```

AutoStart (ASTART) connection parameter

Controls whether a local database server is started if no connection is found.

Syntax

{ **AutoStart** | **ASTART** }={ **YES** | **NO** }

Usage

Anywhere

Default

YES

Remarks

By default, if no database server is found during a connection attempt, and a database file, database name, or the START connection parameter is specified, then a personal database server is started on the same computer. You can turn this behavior off by setting the AutoStart (ASTART) connection parameter to NO

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

in the connection string. The database server is not started automatically if the CommLinks (LINKS) parameter includes TCPIP.

To improve query performance for databases that are started automatically, start the database as soon as possible, even if users are not connecting right away. This allows the cache to warm before queries are executed against the database. See "Using cache warming" [*SQL Anywhere Server - SQL Usage*].

See also

- "Using connection parameters" on page 86
- "Locating a database server" on page 141
- "CommLinks (LINKS) connection parameter" on page 272
- "Elevate connection parameter" on page 286

AutoStop (ASTOP) connection parameter

Controls whether a database is stopped when there are no more open non-HTTP connections.

Syntax

{ **AutoStop** | **ASTOP** }={ **YES** | **NO** }

Usage

Embedded databases

Default

YES

Remarks

By default, any database server that is started from a connection string is stopped when there are no more non-HTTP connections to it. As well, any database that is loaded from a connection string is unloaded when there are no more non-HTTP connections to it. This behavior is equivalent to AutoStop=YES.

If you supply AutoStop=NO, any database that you start in that connection remains running when there are no more non-HTTP connections to it. As a result, the database server remains operational as well.

If the only connection to a database is an HTTP connection, and the database is configured to stop automatically, when the HTTP connection disconnects, the database does not stop automatically. As well, if a database that is configured to stop automatically has an HTTP connection and a command sequence or TDS connection, when the last command sequence or TDS connection disconnects, the database stops, and any HTTP connections are dropped. See "-ga dbeng12/dbsrv12 server option" on page 183 and "AutoStop (ASTOP) connection parameter" on page 269.

The AutoStop (ASTOP) connection parameter is used only if you are connecting to a database that is not currently running. It is ignored if the database is already started.

In .NET applications, you should be careful when using the AutoStop connection parameter. Closing a connection will close it as far as the application is concerned, but active connections remain open when

connection pooling is enabled. As a result the server does not shut down, even though you may expect it to do so.

See also

- "Using connection parameters" on page 86
- "Connection pooling" [*SQL Anywhere Server - Programming*]
- "Starting and stopping databases" on page 36
- "START DATABASE statement" [*SQL Anywhere Server - SQL Reference*]

CharSet (CS) connection parameter

Specifies the character set to be used on this connection.

Syntax

{ **CharSet** | **CS** }={ **NONE** | *character-set* }

Usage

Anywhere

Allowed values

- **NONE** Specifying CharSet=none disables character set conversion for the connection.
- **character-set** Specify the character set label that you want to use. For information about valid character set values, see "Recommended character sets and collations" on page 436.

Default

The local character set.

For information about how the local character set is determined, see "Determining locale information" on page 431.

Remarks

If you supply a value for CharSet, the specified character set is used for the current connection.

When unloading data, you can specify the character set using the CharSet connection parameter.

To avoid lossy character set conversions, setting the CharSet connection parameter is not recommended when using Unicode client APIs. Unicode client APIs include ADO.NET, OLE DB, and the SQL Anywhere JDBC driver. ODBC is also a Unicode client API when the wide (Unicode) functions are used.

See also

- "Using connection parameters" on page 86
- "SACHARSET environment variable" on page 382
- "Understanding the locale character set" on page 418

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Example

The following connection string fragment specifies that Windows Code Page 1252 should be used for the connections:

```
CharSet=windows-1252
```

CommBufferSize (CBSIZE) connection parameter

Sets the maximum size of communication packets.

Syntax

{ **CommBufferSize** | **CBSIZE** }=*size*[**k**]

Usage

Anywhere

Allowed values

- **size** This integer specifies the maximum size of communication packets. The default value is in bytes, but you can use **k** to specify units of kilobytes. The minimum value of CommBufferSize is 500 bytes, and the maximum is 16000 bytes.

Default

If no CommBufferSize value is set, the CommBufferSize is controlled by the setting on the server, which defaults to 7300 bytes on all operating systems except Windows Mobile. On Windows Mobile, the default is 1460 bytes.

Remarks

The protocol stack sets the maximum size of a packet on a network. If you set the CommBufferSize to be larger than that permitted by your network, the communication packets are broken up by the network software. The default size is a multiple of the standard ethernet TCP/IP maximum packet size (1460 bytes).

A larger packet size may improve performance for multi-row fetches and fetches of larger rows, but it also increases memory usage for both the client and the server.

If CommBufferSize is not specified on the client, the connection uses the server's buffer size. If CommBufferSize is specified on the client, the connection uses the CommBufferSize value.

Using the -p database server option to set the CommBufferSize causes all clients that do not specify their own CommBufferSize to use the size specified by the -p database server option.

See also

- "Using connection parameters" on page 86
- "Tuning TCP/IP performance" on page 78
- "-p dbeng12/dbsrv12 server option" on page 212

Example

To set the buffer size to 1460 bytes:

```
...
CommBufferSize=1460
...
```

Alternatively, you can set this parameter by entering its value in the **CommBufferSize** text box on the **Advanced** tab of the **ODBC Configuration For SQL Anywhere** window.

CommLinks (LINKS) connection parameter

Specifies client-side network protocol options.

Note
You should only use the CommLinks (LINKS) connection parameter only if you need to specify TCP/IP options. In most cases, you should use the HOST connection parameter. See "Host connection parameter" on page 291.

Syntax

{ **CommLinks** | **LINKS** }={ [**SharedMemory** | **ShMem**] | **ALL** | [**TCPIP** | **TCP**] } [, ...] *string*

Usage

Anywhere. The CommLinks (LINKS) connection parameter is optional for connections to a personal server, and required for connections to a network server.

Allowed values

CommLinks (LINKS) connection parameter values are case insensitive, and include:

- **SharedMemory (ShMem)** Start the shared memory protocol for same-computer communication. This is the default setting. The client tries shared memory first if it is included in a list of protocols, regardless of the order in which protocols appear.
- **ALL** Attempt to connect using the shared memory protocol first, followed by TCP/IP. Use this setting if you are unsure which communication protocol(s) to use.
- **TCPIP (TCP)** Start the TCP/IP communication protocol. TCP/IP is supported on all operating systems. A personal database server is not started automatically if the CommLinks (LINKS) parameter includes TCPIP.

Each of these values can have additional network protocol options supplied. See "Network protocol options" on page 311.

Default

Use only the shared memory communication protocol to connect.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

If you do not specify a Host connection parameter or a CommLinks (LINKS) connection parameter, the client searches for a database server on the current computer only, and only using a shared memory connection. This is the default behavior, and is equivalent to CommLinks=ShMem. The shared memory protocol is the fastest communication link between a client and database server running on the same computer, as is typical for applications connecting to a personal database server.

For information about securing shared memory connections on Unix, see "Security tips" on page 1116.

If you specify CommLinks=ALL, the client searches for a server using all available communication protocols. Since there may be an impact on performance if you specify CommLinks=ALL, use this setting only when you don't know which protocol to use.

If you specify both TCPIP and SharedMemory, shared memory is attempted first, followed by TCP/IP if the database server cannot be found over shared memory. Shared memory is attempted first, even if TCP/IP is specified first.

The CommLinks (LINKS) connection parameter corresponds to the -x database server option.

If you specify multiple settings for the same connection parameter, then the last value specified is the one that is used. For example, in the following connection string, both shared memory and TCP/IP are specified. In this case, TCP/IP is used because it was specified later in the string.

```
"UID=DBA;PWD=***;Server=demo;START=d:\sa12\bin64\dbsrv12.exe
-x tcpip(port=3277);DBN=demo;DBF=d:\sa12\samples
\demo.db;LINKS=SHMEM;ENC=none;
CommLinks=tcpip(HOST=localhost)"
```

The Host connection parameter is an alias to several existing TCP/IP options. In most cases you should not need to use the LINKS connection parameter. Because the host name is required, but the port number and ServerName connection parameter are optional, there are four possible combinations:

Example	Description	Equivalent LINKS connection parameter connection string
Host=serverhost:1234	This string provides a unique way to find the database server.	LINKS=tcpip(host=serverhost:1234;DoBroadcast=None;Verify=No)
Host=serverhost:1234; ServerName=myserver	This string provides a unique way to find the database server and a server name to verify.	LINKS=tcpip(host=serverhost:1234;DoBroadcast=None;Verify=Yes); ServerName=myserver

Example	Description	Equivalent LINKS connection parameter connection string
Host=serverhost	This connection string fragment does not provide a unique way to find the database server. Because a database server name is not specified, the client does not broadcast to find the database server. The client assumes a port number of 2638 so that it has a unique way to find the database server.	LINKS=tcpip(host=serverhost:2638;DoBroadcast=None;Verify=No)

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Example	Description	Equivalent LINKS connection parameter connection string
Host=serverhost; ServerName=myserver	The host and the database server name are know, but the port number is not. The client sends a UDP packet to host serverhost, port 2638. If the client receives a response, then the response contains the database server's port number. The client makes a TCP connection to that host/port and verifies the database server name.	LINKS=tcpip(host=serverhost;DoBroadcast=Direct;Verify=Yes); ServerName=myserver

See also

- “Host connection parameter” on page 291
- “Network protocol options” on page 311
- “Client/server communications” on page 76
- “-x dbeng12/dbsrv12 server option” on page 236
- “Using connection parameters” on page 86
- “Server name caching for faster connections” on page 144
- “CommLink connection property” on page 623

Example

The following connection string fragment starts the TCP/IP protocol only:

```
CommLinks=tcpip
```

The following connection string fragment starts the shared memory protocol and searches for the database server over shared memory. If the search fails, it then starts the TCP/IP protocol and searches for the server on the local network.

```
CommLinks=tcpip,shmem
```

The following connection string fragment starts the shared memory protocol and searches for the server over shared memory. If the search fails, the TCP protocol is started and it searches for the server on the local network, and the host kangaroo. Note that if the server is found over shared memory, the TCP link is not started.

```
CommLinks=shmem,tcpip(HOST=kangaroo)
```

Compress (COMP) connection parameter

Turns compression on or off for a connection.

Syntax

{ **Compress** | **COMP** }={ **YES** | **NO** }

Usage

Anywhere except with TDS connections. TDS connections (including jConnect) do not support SQL Anywhere communication compression.

Default

NO

Remarks

The packets sent between a SQL Anywhere client and database server can be compressed using the Compress (COMP) connection parameter. Compressing a connection may improve performance under some circumstances. Large data transfers with highly compressible data tend to get the best compression rates.

If a value is not set for the Compress connection parameter, the compression status is controlled by the setting on the database server, which defaults to no compression. If the client and database server settings are different, the client setting applies.

It is recommended that you conduct a performance analysis on the particular network and using the particular application before using communication compression in a production environment.

To enable compression for all remote connections on the database server, use the -pc server option.

Note that same-computer connections over any communication link do not enable compression, even if the -pc option or COMPRESS=YES parameter is used.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Using connection parameters" on page 86
- "-pc dbsrv12 server option" on page 212
- "Adjusting communication compression settings to improve performance" on page 84
- "Use compression carefully" [*SQL Anywhere Server - SQL Usage*]

Example

The following connection string fragment turns packet compression on:

```
Compress=YES
```

The following connection string fragment turns packet compression off:

```
Compress=NO
```

CompressionThreshold (COMPTH) connection parameter

Increases or decreases the size limit at which packets are compressed.

Syntax

{ **CommBufferSize** | **CBSIZE** }=*size*[**k**]

Usage

Anywhere except TDS. Only applies to compressed connections.

Allowed values

- **size** This integer specifies the size limit at which packets are compressed. The default value is in bytes, but you can use **k** to specify units of kilobytes. The minimum value of CommBufferSize is 500 bytes, and the maximum is 16000 bytes. If both the client and database server specify different compression threshold settings, the client setting applies. The minimum supported value is 1 byte, and the maximum supported value is 32767 bytes. Values less than 80 bytes are not recommended.

Default

120

If no CompressionThreshold value is set, the compression threshold value is controlled by the setting on the server, which defaults to 120 bytes.

Remarks

Changing the compression threshold can help performance of a compressed connection by allowing you to only compress packets when compression will increase the speed at which the packets are transferred.

When compression is enabled, individual packets may or may not be compressed, depending on their size. For example, SQL Anywhere does not compress packets smaller than the compression threshold, even if communication compression is enabled. As well, small packets (less than about 100 bytes) usually do not compress at all. Since CPU time is required to compress packets, attempting to compress small packets could actually decrease performance.

Generally speaking, lowering the compression threshold value may improve performance on very slow networks, while raising the compression threshold may improve performance by reducing CPU. However, since lowering the compression threshold value will increase CPU usage on both the client and server, a performance analysis should be done to determine whether changing the compression threshold is beneficial.

See also

- "-pt dbsrv12 server option" on page 213
- "Adjusting communication compression settings to improve performance" on page 84
- "Using connection parameters" on page 86

Example

Connect, with a compression threshold of 100 bytes.

```
CompressionThreshold=100
```

ConnectionName (CON) connection parameter

Names a connection, to make switching to it easier in multi-connection applications.

Syntax

{ **ConnectionName** | **CON** }=*connection-name*

Usage

Anywhere

Allowed values

- **connection-name** This string specifies a name for the particular connection you are making.

Default

No connection name.

Remarks

This connection parameter is optional. You can leave this value unspecified unless you are going to establish more than one connection, and switch between them.

The connection name is not the same as the data source name.

See also

- "Using connection parameters" on page 86
- "SET CONNECTION statement [Interactive SQL] [ESQL]" [*SQL Anywhere Server - SQL Reference*]
- "Name connection property" on page 632

Example

Connect to a database, naming the connection first-con:

```
CON=first-con
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

ConnectionPool (CPOOL) connection parameter

Controls the behavior of client connection pooling.

Syntax

ConnectionPool={ **NO** | **YES** [([**Timeout=***timeout-sec*;] [**MaxCached=***max-cached-conn*])] }

Usage

All platforms except Windows Mobile and non-threaded Unix clients.

Allowed values

- **timeout-sec** The idle timeout period, in seconds, of the connection pool. The default value is 60 seconds. Cached connections that are not reused within the time specified by the *timeout-sec* value are disconnected and are no longer available for reuse.

- **max-cached-conn** The maximum number of cached connections from each application. The default value is five connections. A connection is cached if it is disconnected and the maximum number of connections specified by the *max-cached-conn* value has not been reached. The connection is reinitialized, and the cached connection remains connected to the database server even though the application has disconnected it. For information about cleanup and reinitialization tasks that are performed, see "SQL Anywhere connection pooling" on page 136.

Default

YES

Remarks

Connection pooling may improve the performance of applications that make multiple, brief connections to the database server. When a connection is disconnected it is automatically cached and may be reused when the application reconnects. For a connection to be pooled, the connection name can be different, but all other connection parameters must be identical.

See also

- "SQL Anywhere connection pooling" on page 136
- "Using connection parameters" on page 86
- "ConnPoolCachedCount database property" on page 663
- "ConnPoolHits database property" on page 663
- "ConnPoolMisses database property" on page 663

Example

The following connection string fragment turns connection pooling off.

```
ConnectionPool=NO;
```

The following connection string fragment turns connection pooling on with a maximum of 10 cached connections.

```
CPOOL=YES(MaxCached=10);
```

DatabaseFile (DBF) connection parameter

Indicates which database file you want to load and connect to when starting a database that is not running.

Syntax

{ **DatabaseFile** | **DBF** }=*filename*

Usage

Embedded databases

Allowed values

- **filename** This string specifies the path and file name of a database that is already running.
  - If the file name does not include an extension, SQL Anywhere looks for a file with the *.db* extension.
  - The path of the file is relative to the working directory of the database server. If you start the database server from a command prompt, the working directory is the directory that you are in when entering the command. If you start the database server from an icon or shortcut, it is the working directory that the icon or shortcut specifies. It is recommended that you supply a complete path and file name.
  - If you specify both the database file and the database name, an attempt is made to connect to a running database with the specified name (the database file is ignored), and if that fails, an attempt is made to start a database automatically using both the database file and database name. The database server is not started automatically if the CommLinks (LINKS) parameter includes TCPIP.
  - UNC file names are supported.

    For more information about using UNC file names, see "The SQL Anywhere database server" on page 147.

Default

There is no default setting.

Remarks

If the database you want to connect to is not running, use the DatabaseFile (DBF) connection parameter so the database can be started.

It is recommended that deployed applications specify a database server name using the ServerName (Server) parameter when attempting to start a database file automatically if it is not running. Otherwise, the application may connect to a different database server than intended. For example, the database server could connect to a different version of the SQL Anywhere server that is part of an embedded application and already running.

Caution
The database file must be on the same computer as the database server. Starting a database file that is located on a network drive can lead to file corruption.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "-gd dbeng12/dbsrv12 server option" on page 185
- "CommLinks (LINKS) connection parameter" on page 272
- "DatabaseName (DBN) connection parameter" on page 282
- "Using connection parameters" on page 86
- "Connecting to an embedded database" on page 131

Example

The DatabaseFile (DBF) connection parameter in the following example loads and connects to the sample database, *demo.db*:

```
DBF=samples-dir\demo.db
```

For information about *samples-dir*, see "Samples directory" on page 392.

The following two examples assume that you have started a database file named *cities.db*, and renamed the database Kitchener as follows:

```
dbeng12 cities.db -n Kitchener
```

To successfully start and connect to a database and name it Kitchener:

```
DBN=Kitchener;DBF=cities.db
```

Specifying `DBF=cities.db` would fail to connect to the running database named Kitchener.

DatabaseKey (DBKEY) connection parameter

Starts an encrypted database with a connect request.

Syntax

{ **DatabaseKey** | **DBKEY** }=*key*

Usage

Anywhere

Allowed values

- **key** The encryption key is a string, including mixed cases, numbers, letters, and special characters. Database keys cannot include leading spaces, trailing spaces, or semicolons.

Default

None

Remarks

You must specify this parameter when you start an encrypted database with a connect request. You do not need to specify this parameter if you are connecting to an encrypted database that is already running.

If you want to secure communication packets between client applications and the database server, use the -ec server option and transport-layer security. See "Transport-layer security" on page 1143.

See also

- "Configuring client applications to use transport-layer security" on page 1154
- "-ec dbeng12/dbsrv12 server option" on page 176
- "-ek dbeng12/dbsrv12 database option" on page 256
- "-ep dbeng12/dbsrv12 server option" on page 179
- "-es dbeng12/dbsrv12 server option" on page 180
- "Using connection parameters" on page 86
- "Encryption (ENC) connection parameter" on page 287

Example

The following fragment illustrates the use of the DatabaseKey (DBKEY) connection parameter:

```
"UID=DBA;PWD=sql;Server=myeng;DBKEY=V3moj3952B;DBF=samples-dir\demo.db"
```

DatabaseName (DBN) connection parameter

Identifies a loaded database to which a connection needs to be made when connecting to a database that is already running.

Syntax

{ **DatabaseName** | **DBN** }=*database-name*

Usage

Running local databases or network database servers

Allowed values

- **database-name** This string specifies the name of a database that is already running on a database server.

Default

There is no default setting.

Remarks

If you want to connect to a database that is not running, use the DatabaseFile (DBF) parameter.

Whenever a database is started on a server, it is assigned a database name, either by the administrator using the -n option, or by the server using the base of the file name with the extension and path removed.

You can only use the database name utility_db to connect to the SQL Anywhere utility database. See "Using the utility database" on page 28.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Note
The DatabaseName (DBN) connection parameter is recommended for naming databases, rather than using the DatabaseSwitches (DBS) connection parameter with the -n option.

If the database you want to connect to is already running, you should specify the database name rather than the database file.

A connection only occurs if the name of the running database matches the name that is specified in the DatabaseName (DBN) parameter.

Note
If you specify both the database file and the database name, an attempt is made to connect to a running database with the specified name (the database file is ignored), and if that fails, an attempt is made to start a database automatically using both the database file and database name.

See also

- "Using connection parameters" on page 86
- "DatabaseName (DBN) protocol option" on page 319
- "DatabaseSwitches (DBS) connection parameter" on page 283

Example

To start a database file named *cities.db* and rename the database Kitchener, you can use the following command:

```
dbeng12 cities.db -n Kitchener
```

Assuming you have run the above command, you can successfully connect to the running database named Kitchener as follows:

```
DBN=Kitchener
```

Alternatively, you could use the following to successfully connect to the running database named Kitchener:

```
DBN=Kitchener;DBF=cities.db
```

However, specifying the following would fail to connect to the database named Kitchener:

```
DBF=cities.db
```

DatabaseSwitches (DBS) connection parameter

Provides database-specific options when starting a database.

Syntax

{ **DatabaseSwitches** | **DBS** }=*database-server-options*

Usage

Connecting to a database server when the database is not loaded. This connection parameter starts a database server automatically with the specified database and options if a database server is not running.

Allowed values

- **database-server-options** This string specifies options that apply to the database file. See "Database options" on page 251.

  Database server options must be supplied using the StartLine connection parameter. See "StartLine (START) connection parameter" on page 308.

Default

No options.

Remarks

You should supply DatabaseSwitches only if you are connecting to a database that is not currently running. When the server starts the database specified by DatabaseFile, the server uses the supplied DatabaseSwitches to determine startup options for the database.

Note
The DatabaseName (DBN) connection parameter is recommended for naming databases, rather than using the DatabaseSwitches (DBS) connection parameter with the -n option.

See also

- "The SQL Anywhere database server" on page 147
- "Using connection parameters" on page 86
- "DatabaseName (DBN) connection parameter" on page 282

Example

The following command, entered on one line at a command prompt, connects to the default database server, loads the database file *demo.db* (DatabaseFile (DBF) connection parameter), names it my-db (DatabaseName (DBN) connection parameter) and starts it in read-only mode (-r option).

```
dbisql -c "UID=DBA;PWD=sql;DBF=samples-dir\demo.db;DBN=my-db;DBS=-r"
```

For information about *samples-dir*, see "Samples directory" on page 392.

DataSourceName (DSN) connection parameter

Tells the ODBC driver manager or embedded SQL library where to look in the registry or the system information file (named *.odbc.ini* by default) to find ODBC data source information.

Syntax

{ **DataSourceName** | **DSN** }=*data-source-name*

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Usage

Anywhere

Allowed values

- **data-source-name** This string specifies the name of the ODBC data source that contains connection in formation for your database.

Default

There is no default data source name.

Remarks

It is common practice for ODBC applications to send only a data source name to ODBC. The ODBC driver manager and ODBC driver locate the data source, which contains the remainder of the connection parameters.

In SQL Anywhere, embedded SQL applications can also use ODBC data sources to store connection parameters.

See also

- "FileDataSourceName (FILEDSN) connection parameter" on page 290
- "Using connection parameters" on page 86
- "Using ODBC data sources on Unix" on page 105
- "Creating ODBC data sources" on page 98

Example

The following parameter uses a data source name:

```
DSN=My Database
```

DisableMultiRowFetch (DMRF) connection parameter

Turns off multi-row fetches across the network.

Syntax

{ **DisableMultiRowFetch** | **DMRF** }={ **YES** | **NO** }

Usage

Anywhere

Default

NO

Remarks

By default, when the database server gets a simple fetch request, the application asks for extra rows. You can disable this behavior by setting this parameter to YES. See "Using cursors in procedures and triggers" [*SQL Anywhere Server - SQL Usage*].

Setting the DisableMultiRowFetch (DMRF) connection parameter to YES is equivalent to setting the prefetch database option to Off. See "Prefetching rows" [*SQL Anywhere Server - Programming*].

See also

- "Using connection parameters" on page 86
- "prefetch option" on page 573

Example

The following connection string fragment prevents prefetching:

```
DMRF=YES
```

Elevate connection parameter

Elevates automatically started database server executables on Windows Vista.

Syntax

Elevate={ YES | NO }

Usage

Windows Vista only

Default

NO

Remarks

You can specify ELEVATE=YES in your connection string so that automatically started database server executables are elevated. This allows non-elevated client processes to start elevated servers automatically, which is necessary on Windows Vista because non-elevated servers cannot use AWE memory. This parameter is ignored if the database server is not started automatically. You must specify the -cw option when starting the database server command to use an AWE cache.

See also

- "-cm dbeng12/dbsrv12 server option" on page 167
- "-cw dbeng12/dbsrv12 server option (deprecated)" on page 171

Example

The following connection string fragment causes database servers that are started automatically to be elevated on Windows Vista so that they can use an AWE cache:

```
"Elevate=YES;START=dbeng12 -cw"
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

EncryptedPassword (ENP) connection parameter

Provides a password, stored in an encrypted fashion in a data source.

Syntax

{ **EncryptedPassword** | **ENP** }=*password*

Usage

Anywhere

Allowed values

- **password** This string is an encrypted password for the database you are connecting to.

Default

None

Remarks

Every user of a database has a password. The password must be supplied for the user to be allowed to connect to the database.

The EncryptedPassword (ENP) connection parameter is used to specify an encrypted password. An application may include the encrypted password in the connection string. If both the Password (PWD) connection parameter and the EncryptedPassword (ENP) connection parameter are specified, the Password (PWD) connection parameter takes precedence.

Caution
When creating a data source, it is recommended that you do not include the encrypted password as part of the definition. Although both the **ODBC Configuration For SQL Anywhere** window in the Windows **ODBC Data Source Administrator** and the SQL Anywhere Data Source utility (dbdsn) have this capability, including this information poses a security risk.

You can encrypt a password that is stored in an ODBC data source by using the -pe option with the Data Source utility (dbdsn). See "Data Source utility (dbdsn)" on page 779.

On Unix, data source information is stored in a system information file (named *.odbc.ini* by default).

For information about how this system information file is located, see "Using ODBC data sources on Unix" on page 105.

See also

- "Using connection parameters" on page 86
- "Password (PWD) connection parameter" on page 302

Encryption (ENC) connection parameter

Encrypts packets sent between the client application and the database server using transport-layer security or simple encryption.

Syntax

{ **Encryption** | **ENC** }={ **NONE**
| **SIMPLE**
| **TLS(TLS_TYPE=***cipher*;
[**FIPS=**{ **Y** | **N** };]
TRUSTED_CERTIFICATES=*public-certificate*;
[**CERTIFICATE_COMPANY=***organization*;]
[**CERTIFICATE_NAME=***common-name*;]
[**CERTIFICATE_UNIT=***organization-unit*] **)**

Usage

TLS: supported for TCP/IP only

NONE or SIMPLE: anywhere

Allowed values

- **NONE** Accepts communication packets that are not encrypted.

- **SIMPLE** Accepts communication packets that are encrypted with simple encryption supported on all platforms and on previous versions of SQL Anywhere. Simple encryption does not provide server authentication, strong elliptic-curve or RSA encryption, or other features of transport-layer security.

  If the database server accepts simple encryption, but does not accept no encryption, then any non-TDS connection attempts using no encryption automatically use simple encryption.

  Starting the database server with `-ec SIMPLE` tells the database server to accept only connections using simple encryption. TLS connections (ECC, RSA, RSA FIPS) fail, and connections requesting no encryption use simple encryption.

  Starting the database server with `-ec SIMPLE,TLS( TLS_TYPE=ECC;... )` tells the database server to accept only connections with ECC TLS encryption or simple encryption. Both RSA and RSA FIPS connections fail, and connections requesting no encryption use simple encryption.

- ***cipher*** can be **RSA** or **ECC** for RSA and ECC encryption, respectively. For FIPS-approved RSA encryption specify **TLS_TYPE=RSA;FIPS=Y**. RSA FIPS uses a separate approved library, but is compatible with servers specifying RSA with SQL Anywhere 9.0.2 or later.

  The connection fails if the cipher does not match the encryption (RSA or ECC) used to create your certificates.

  The client can use the following arguments to verify the field values in the server's public certificate:

  - trusted_certificates
  - certificate_company
  - certificate_unit
  - certificate_name

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For more information about verifying certificate fields for server authentication, see "Verifying certificate fields" on page 1154.

Default

NONE

Remarks

You can use this parameter if you want to secure communications between client applications and the database server using transport-layer security or simple encryption. See "Transport-layer security" on page 1143.

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

For more information about using digital certificates, see "Creating digital certificates" on page 1146.

You can use the CONNECTION_PROPERTY system function to retrieve the encryption settings for the current connection:

```
SELECT CONNECTION_PROPERTY ( 'Encryption' );
```

The function returns one of five values: None, Simple, ecc_tls, rsa_tls, or rsa_tls_fips depending which type of encryption is being used by the connection.

See "CONNECTION_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].

See also

- "Configuring client applications to use transport-layer security" on page 1154
- "-ec dbeng12/dbsrv12 server option" on page 176
- "-ek dbeng12/dbsrv12 database option" on page 256
- "-ep dbeng12/dbsrv12 server option" on page 179
- "-es dbeng12/dbsrv12 server option" on page 180
- "Using connection parameters" on page 86
- "DatabaseKey (DBKEY) connection parameter" on page 281
- "certificate_company protocol option" on page 315
- "certificate_name protocol option" on page 316
- "certificate_unit protocol option" on page 317
- "trusted_certificates protocol option" on page 339
- "Encryption connection property" on page 626

Example

The following connection string fragment connects to a database server using transport-layer security and elliptic-curve encryption:

```
"Host=myhost;Server=myserver;ENCRYPTION=tls(tls_type=ecc;trusted_certificates
=eccroot.crt)"
```

The following connection string fragment connects to a database server using transport-layer security and RSA encryption:

```
"Host=myhost;Server=myserver;ENCRYPTION=tls(tls_type=rsa;fips=n;trusted_certi
ficates=rsaroot.crt)"
```

The following connection string fragment connects to a database server using simple encryption:

```
"Host=myhost;Server=myserver;ENCRYPTION=simple"
```

EngineName (ENG) connection parameter (deprecated)

This connection parameter is deprecated. Use the ServerName (Server) connection parameter instead. See "ServerName (Server) connection parameter" on page 306.

FileDataSourceName (FILEDSN) connection parameter

Tells the client library there is an ODBC file data source holding information about the database to which you want to connect.

Syntax

{ **FileDataSourceName** | **FILEDSN** }=*file-data-source-name*

Usage

Anywhere

Allowed values

- **file-data-source-name** This string specifies the name of the file data source that contains connection in formation for your database.

Default

There is no default name.

Remarks

File data sources hold the same information as ODBC data sources stored in the registry. File data sources can be easily distributed to end users so that connection information does not have to be reconstructed on each computer.

Both ODBC and embedded SQL applications can use file data sources.

See also

- "Using file data sources on Windows" on page 103
- "DataSourceName (DSN) connection parameter" on page 284
- "Using connection parameters" on page 86
- "Using file data sources on Windows" on page 103

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Host connection parameter

Accepts a host name or IP address and optional port number that tells the client where to find the database server.

Syntax

Host={ *hostname* | *ip-address* }[**:***port-number*] ...

Usage

Anywhere. The Host connection parameter is optional for connections to a personal server, and recommended for connections to a network server.

Allowed values

- **hostname** The name of the computer where the database server is running. The list of host values is a comma-separated list and it can optionally include a port number (separated by a colon). You can use **localhost** to identify the current computer.
- **ip-address** This string must be specified in the form of an IP address and it can optionally include a port number (separated by a colon). The list of IP addresses is a comma-separated list.

  If you specify an IPv6 address with a port number the address must include brackets before the port. For example, `HOST=(fd77:ab34:2238::3894):8933`, where 8933 is the port number.
- **port-number** The port number used by the database server. The default port number is 2638.

Default

None

Remarks

The Host connection parameter specifies a hostname (or IP address) and optional port number that tell the client where a database server is running.

When you use the Host parameter, no UDP packets are sent if enough information is given to uniquely identify the server (a host name and a port number). If neither a port number nor database server name is given, the port number is assumed to be 2638 and the client does not perform a broadcast. However, if the client has a host name and database server name but no port number, it sends a UDP packet to port 2638 on the specified host to find the port number.

The Host connection parameter can take a list of host names, separated by commas. The hosts are attempted the in order given, with the exception that if the address stored in the database server address cache matches any one of the addresses specified by the Host connection parameter, then that one is tried first.

When the Host connection parameter is used, the ServerName connection parameter is optional. If the database server name is given, it is verified once the connection is made (the client ensures that the database server it is connected to has the specified name). It is recommended that you always use the ServerName parameter, particularly for embedded applications. See "ServerName (Server) connection parameter" on page 306.

Note
If you specify the Host connection parameter, you cannot specify the CommLinks (LINKS) connection parameter. You should only use the CommLinks (LINKS) connection parameter only if you need to specify TCP/IP options. See "CommLinks (LINKS) connection parameter" on page 272.

See also

- "Connecting to SQL Anywhere databases" on page 87
- "CommLinks (LINKS) connection parameter" on page 272

Example

If you know that a database server named sales is running on a computer called elora on the default port number, then you can use the following connection string to connect to the database server:

```
UID=DBA;PWD=sql;Server=sales;Host=elora:2638
```

If you do not know the port number that the database server is running on, use the following connection string to connect to the sales database server running on the computer named elora:

```
UID=DBA;PWD=sql;Server=sales;Host=elora
```

ForceStart (FORCE) connection parameter

Starts a database server without attempting to connect to one.

Syntax

{ **ForceStart** | **FORCE** }={ **YES** | **NO** }

Usage

Only with the db_start_engine function.

Default

NO

Remarks

By setting ForceStart=YES, the db_start_engine function starts a server without attempting to connect to one, even if there is one already running.

See also

- "db_start_engine function" [*SQL Anywhere Server - Programming*]
- "Using connection parameters" on page 86

Idle connection parameter

Specifies a connection's idle timeout period.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

Idle=*timeout-value*

Usage

Anywhere except with TDS connections. TDS connections (including jConnect) ignore the SQL Anywhere Idle (IDLE) connection parameter.

Allowed values

- **timeout-value** The connection's idle timeout period, in minutes. The minimum value for the Idle connection parameter is 1 minute, and the maximum supported value is 32767 minutes. If you specify 0, idle timeout checking is turned off for the connection.

Default

240 minutes (TCP/IP)

0 (shared memory)

Remarks

The Idle (IDLE) connection parameter applies only to the current connection. You can have multiple connections on the same database server set to different timeout values.

If no connection idle timeout value is set, the idle timeout value is controlled by the setting on the database server. If a conflict between timeout values occurs, the connection timeout value supersedes a specified or unspecified database server timeout value.

See also

- "-ti dbeng12/dbsrv12 server option" on page 226
- "Using connection parameters" on page 86
- "Adjusting timeout values" on page 1005
- "IdleTimeout connection property" on page 628

Example

The following connection string fragment sets the timeout value for this connection to 10 minutes:

```
"Host=myhost;IDLE=10"
```

Integrated (INT) connection parameter

Specifies whether an integrated login can be attempted.

Syntax

{ **Integrated** | **INT** }={ **YES** | **NO** }

Usage

Anywhere

Allowed values

- **YES** An integrated login is attempted. If the connection attempt fails and the login_mode option is set to Standard,Integrated, a standard login is attempted.
- **NO** This is the default setting. No integrated login is attempted.

Default

NO

Remarks

For a client application to use an integrated login, the server must be running with the login_mode database option set to a value that includes Integrated.

See also

- "login_mode option" on page 547
- "Using connection parameters" on page 86
- "Using Windows integrated logins" on page 108

Example

The following data source fragment uses an integrated login:

```
INT=YES
```

Kerberos (KRB) connection parameter

Specifies whether Kerberos authentication can be used when connecting to the database server.

Syntax

{ **Kerberos** | **KRB** }= { **YES** | **NO** | **SSPI** | *GSS-API-library-file* }

Usage

All platforms except Windows Mobile.

Allowed values

- **YES** A Kerberos authenticated login is attempted.
- **NO** No Kerberos authenticated login is attempted. This is the default.
- **SSPI** A Kerberos authenticated login is attempted, and the built-in Windows SSPI interface is used instead of a GSS-API library. SSPI can only be used on Windows platforms, and it cannot be used with a Key Distribution Center (KDC) other than the Domain Controller Active Directory KDC. If your Windows client computer has already logged in to a Windows domain, SSPI can be used without needing to install or configure a Kerberos client.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Note
SSPI can only be used by SQL Anywhere clients in the Kerberos connection parameter. SQL Anywhere database servers cannot use SSPI—they need a supported Kerberos client other than SSPI.

- **GSS-API-library-file** A Kerberos authenticated login is attempted, and this string specifies the file name of the Kerberos GSS-API library (or shared object on Unix). This is only required if the Kerberos client uses a different file name for the Kerberos GSS-API library than the default, or if there are multiple GSS-API libraries installed on the computer.

Default

NO

Remarks

The UserID and Password connection parameters are ignored when using a Kerberos authenticated login.

To use Kerberos authentication, a Kerberos client must already be installed and configured (nothing needs to be done for SSPI), the user must have already logged in to Kerberos (have a valid ticket-granting ticket), and the database server must have enabled and configured Kerberos authenticated logins.

See also

- "-kl dbeng12/dbsrv12 server option" on page 201
- "-kr dbeng12/dbsrv12 server option" on page 202
- "-krb dbeng12/dbsrv12 server option" on page 203
- "Kerberos authentication" on page 116
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "Use SSPI for Kerberos logins on Windows" on page 122

Example

```
Kerberos=YES
Kerberos=SSPI
Kerberos=c:\Program Files\MIT\Kerberos\bin\gssapi32.dll
```

Language (LANG) connection parameter

Specifies the language of the connection.

Syntax

{ **Language** | **LANG** }=*language-code*

Usage

Anywhere

Allowed values

- **language-code** The two-letter combination representing a language. For example, specifying LANG=DE sets the default language to German.

For more information about language codes, see "Understanding the locale language" on page 417.

Default

The language specified by (in order) the SALANG environment variable, the dblang utility, or the installer.

Remarks

This connection parameter establishes the language for the connection. Any errors or warnings from the server are delivered in the specified language, assuming that the server supports the language.

If no language is specified, the default language is used. The default language is the language specified by, in order, the SALANG environment variable, the dblang utility, or the installer.

This connection parameter only affects the connection. Messages returned from SQL Anywhere tools and utilities appear in the default language, while the messages returned from the server appear in the connection's language.

See also

- "Using connection parameters" on page 86

LazyClose (LCLOSE) connection parameter

Controls whether cursor requests are queued until the next request or performed immediately. Queuing close cursor requests saves a round trip and improves performance.

Syntax

{ **LazyClose** | **LCLOSE** }={ **YES** | **NO** | **AUTO** }

Usage

Anywhere

Allowed values

- **YES** Always queue the cursor close request, which saves a round trip, but can cause locks and other resources to be held after the cursor is closed by the client. The cursor close is performed when the next request is sent to the database server on the same connection. Any isolation level 1 cursor stability locks still apply to the cursor while the `CLOSE` *`cursor-name`* database request is queued.
- **NO** Close the cursor immediately.
- **AUTO** Queue the cursor close request and save a round trip, only when doing so doesn't change how long locks or significant server resources are held. If the cursor uses isolation level 1 cursor stability locks, or could consume significant server resources that are not released until the cursor is closed, then the cursor is closed immediately. A query that requires a work table is an example of a cursor that can consume significant server resources.

Default

AUTO

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

When this connection parameter is set to YES or AUTO, cursors are not closed until the next database request.

Enabling this option can improve performance, if your network exhibits poor latency or your application sends many cursor open and close requests.

See also

- "Using connection parameters" on page 86
- "Reduce requests between client and server" [*SQL Anywhere Server - SQL Usage*]

LivenessTimeout (LTO) connection parameter

Controls the shutdown of connections when they are no longer intact.

Syntax

{ **LivenessTimeout** | **LTO** }=*timeout-value*

Usage

Network server only.

All platforms except non-threaded Unix applications.

Allowed values

- **timeout-value** The connection's liveness timeout period, in seconds. The minimum value for the LivenessTimeout connection parameter is 30 seconds, and the maximum value is 32767 seconds. If you specify 0, liveness timeout checking is turned off for the connection. Any non-zero value less than the minimum value is reset to the minimum value. For example, a connection string containing "LivenessTimeout=5" uses "LivenessTimeout=30". If no LivenessTimeout value is set, the LivenessTimeout is controlled by the setting on the server, which defaults to 120 seconds.

Default

None

Remarks

A **liveness packet** is sent periodically across a client/server TCP/IP communication protocol to confirm that a connection is intact. If the client runs for the LivenessTimeout period without detecting a liveness request or response packet, the communication is ended.

Liveness packets are sent when a connection has not sent any packets for between one third and two thirds of the LivenessTimeout value.

When there are more than 200 connections to a server, the server automatically calculates a higher LivenessTimeout value based on the stated LivenessTimeout value. This enables the server to handle a large number of connections more efficiently.

Alternatively, you can set this parameter by entering its value in the **LivenessTimeout** text box on the **Advanced** tab of the **ODBC Configuration For SQL Anywhere** window.

See also

- "Using connection parameters" on page 86
- "-tl dbeng12/dbsrv12 server option" on page 226
- "LivenessTimeout connection property" on page 629

Example

The following connection string fragment sets a LivenessTimeout value of 10 minutes:

```
LTO=600
```

LogFile (LOG) connection parameter

Sends client error messages and debugging messages to a file.

Syntax

{ **LogFile** | **LOG** }=*filename*

Usage

Anywhere

Allowed values

- **filename** This string specifies the name of the file where client error messages and debugging messages are saved. If the file name does not include a path, it is relative to the current working directory of the client application.

Default

No log file.

Remarks

The LogFile (LOG) connection parameter is connection-specific, so from a single application you can set different LogFile arguments for different connections.

Typical log file contents are as follows:

```
Tue Jun 08 2010 13:39:12
13:39:12 Attempting to connect using:
UID=DBA;PWD=********;DBF='C:\Documents and Settings\All Users\Documents\SQL
Anywhere 12\Samples\demo.db';
ServerName=demo12;START='C:\Program Files\SQL Anywhere
12\Bin32\dbeng12.exe';CON=SQL_DBC_48055888;ASTOP=YES;LOG=c:\logs\t
13:39:12 Attempting to connect to a running server...
13:39:12 Trying to start SharedMemory link ...
13:39:12     SharedMemory link started successfully
13:39:12 Attempting SharedMemory connection (no sasrv.ini cached address)
13:39:12 Connected to server over SharedMemory
13:39:12 Connected to SQL Anywhere Server version 12.0.0.2413
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
13:39:12 Application information:
13:39:13 IP=10.25.99.117;HOST=DCHAMPAG-D630;OSUSER=user1;OS='Windows XP
Build 2600 Service Pack 3';
EXE='C:\Program Files\SQL Anywhere
12\bin32\dbisql.exe';PID=0x112c;THREAD=0x16fc;VERSION=12.0.0.2413;
API=iAnywhereJDBC;TIMEZONEADJUSTMENT=-240
13:39:13 Connected to the server, attempting to connect to a running
database...
13:39:14 [    3] Connected to database successfully
13:39:15 [    3] The number of prefetch rows has been reduced to 16 due to the
prefetch buffer
13:39:15 [    3] limit. Increasing the PrefetchBuffer connection parameter may
improve performance.
13:39:29 [    3] Disconnected from server
```

See also

- "Using connection parameters" on page 86
- "LogFile (LOG) protocol option" on page 328

Example

The following command line starts Interactive SQL, connecting to the sample database with a LogFile (LOG) connection parameter:

```
dbisql -c "DSN=SQL Anywhere 12 Demo;LOG=c:\logs\test.txt"
```

NewPassword (NEWPWD) connection parameter

Allows users to change passwords, even if they have expired, without DBA intervention.

Syntax

{ **NewPassword** | **NEWPWD** }={ *password-string* | ***** }

Usage

Anywhere. The client library prompting for a new password is only supported on Microsoft Windows.

Allowed values

- **password-string** If the user provides a new password, the database server authenticates the user ID and password and attempts to change the password before the login_procedure option is called. This process allows the user to change an expired password without the involvement of a DBA. If you have set the verify_password_function option, the new password is verified. If you are authenticating with an Integrated or Kerberos login, the original password is not validated and the database server ignores the new password value and the password is not changed.

- ***** On Microsoft Windows, if you use the special value *, the client library prompts for a new password during a connection attempt only if the existing password has expired. The user must provide their existing password, provide their new password, and confirm their new password. When the user completes the fields and clicks **OK**, the old password is authenticated and the database server attempts to change the password. If you have set the verify_password_function option, the new password is verified. The process of verifying if a user's password has expired, prompting for a

password, and authenticating and changing the password occurs with a single connect call to the client library.

Default

The password is not changed, and the client library does not prompt for a new password.

Remarks

This connection parameter is very effective when you implement a login policy using the password_life_time or password_expiry_on_next_login options. Alternatively, you can implement a password expiry policy by having the login_procedure signal the `Password has expired` error.

A user receives a `Password has expired error` if their environment does not support password prompting. In a Microsoft Windows environment, the prompt window might not correctly prevent interaction with the calling application's window (it may not be modal or have the correct parent window) if the calling application has multiple top-level windows or if the application's top level windows are minimized.

In a Windows environment, if you use the ODBC SQLDriverConnect function and the DriverCompletion argument is anything other than SQL_DRIVER_NOPROMPT, the connection prompts for a new password if the password has expired. The connection might prompt for a new password in OLE DB when the DBPROP_INIT_PROMPT property is anything other than DBPROMPT_NOPROMPT. Both cases function as if the NewPassword=***** connection parameter was specified.

See also

- “GRANT statement” [*SQL Anywhere Server - SQL Reference*]
- “login_procedure option” on page 549
- “verify_password_function option” on page 614
- “post_login_procedure option” on page 571

Example

The following connection string changes the password of user Test1 when they connect:

```
"UID=Test1;PWD=welcome;NEWPWD=hello"
```

In a Windows environment, the following connection string prompts the user Test1 for a new password when the existing password expires:

```
"UID=Test1;PWD=welcome;NEWPWD=*"
```

NodeType (NODE) connection parameter

Separately licensed component required
Read-only scale-out and database mirroring each require a separate license. See “Separately licensed components” [*SQL Anywhere 12 - Introduction*].

Controls load balancing among database servers involved in read-only scale-out.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

{ **NodeType** | **NODE** }={ **DIRECT** | **PRIMARY** | **COPY** }

Usage

Network server only.

Allowed values

- **DIRECT** This is the default setting. When the NodeType connection parameter is set to DIRECT, the database server accepts the connection without performing load balancing or redirection.
- **PRIMARY** If the NodeType connection parameter is set to PRIMARY and you have connected to the primary server, the connection is accepted. If you have connected to the mirror server or a copy node, the database server redirects the connection to the primary server.
- **COPY** When the NodeType connection parameter is set to COPY, the database server examines the copy nodes in its own branch (including itself if it is not the root node) and chooses the copy node with the lightest load. If the database server does not choose itself, it redirects the client to the chosen database server.

Default

DIRECT

Remarks

This connection parameter is used by client applications to control load balancing among database servers involved in read-only scale-out. Clients specify this connection parameter to indicate what type of database server they want to connect to. The database server that the client initially connects to decides which database server should handle the connection, and if necessary, redirects the connection to the appropriate database server by returning the address of the database server. The client is disconnected from the original database server, and then connects to the appropriate database server.

When you specify COPY, the load balancing occurs on the branch that contains the database server the client connected to. A branch consists of that database server and any of its children. This feature may be useful for performing load balancing among geographically separate servers.

If a client connection is redirected to a different database server, the value specified in the ServerName (Server) connection parameter and the value of the Name database server property do not match. See "Name server property" on page 653.

When NodeType is specified, the application typically connects to the root node, and that database server determines which copy node is least heavily loaded. The connection is then redirected to that copy node. If the application makes and drops several such connections within a short period of time, the connection is pooled and the root node is not asked which copy server to use. Using connection pooling reduces the load on the root node, but may not give expected behavior. The application can specify that its connections cannot be pooled to ensure that the root server determines which copy node to connect on each connection.

See also

- "Introduction to database mirroring" on page 945
- "SQL Anywhere read-only scale-out" on page 980

Example

The primary and mirror servers are located in Waterloo, Ontario. Clients running in Japan may want to set up a group of copy nodes in Japan to reduce connection latency. If a branch of copy nodes in Japan was a direct child of the root node, then other children could be added beneath the copy node in Japan. Clients would connect to the hub and specify NodeType=COPY. Japanese local, read-only connections would then be load-balanced among the local database servers, and no client would ever connect to database servers elsewhere. If a client needed to make changes to the database, they would change their connection string to use NodeType=PRIMARY.

Password (PWD) connection parameter

Provides a password for a connection.

Syntax

{ **Password** | **PWD** }=*password-string*

Usage

Anywhere

Allowed values

- **password-string** Passwords have a maximum length of 255 bytes and are case sensitive. Passwords cannot include leading spaces, trailing spaces, or semicolons.

Default

No password provided.

Remarks

Every user of a database has a password. The password must be supplied for the user to be allowed to connect to the database.

The Password (PWD) connection parameter is not encrypted. An application can include the password in the connection string. If both the Password (PWD) connection parameter and the EncryptedPassword (ENP) connection parameter are specified, the Password (PWD) connection parameter takes precedence.

Caution
When creating a data source, it is recommended that you do not include the password as part of the definition. Although both the **ODBC Configuration For SQL Anywhere** window in the Windows **ODBC Data Source Administrator** and the SQL Anywhere Data Source utility (dbdsn) have this capability, including this information poses a security risk.

On Unix, data source information is stored in a system information file (named *.odbc.ini* by default).

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For more information about how this system information file is located, see "Using ODBC data sources on Unix" on page 105.

See also

- "Setting a password" on page 451
- "Increasing password security" on page 1118
- "EncryptedPassword (ENP) connection parameter" on page 287
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "Case sensitivity" [*SQL Anywhere Server - SQL Usage*]
- "Using connection parameters" on page 86

Example

The following connection string fragment supplies the user ID DBA and password sql.

```
UID=DBA;PWD=sql
```

PrefetchBuffer (PBUF) connection parameter

Sets the maximum amount of memory for buffering rows, in bytes.

Syntax

{ **PrefetchBuffer** | **PBUF** }=*buffer-size*[**k** | **m**]

Usage

Anywhere

Allowed values

- **buffer-size** The value is in bytes, but you can use **k** or **m** to specify units of kilobytes or megabytes, respectively. This connection parameter accepts values between 64 KB and 8 MB.

  For compatibility with previous versions, if a value less than 16384 is specified, it is interpreted as kilobytes.

  Using kilobytes without the k suffix in the PrefetchBuffer connection parameter is deprecated. See "PrefetchRows (PROWS) connection parameter" on page 305.

Default

512 KB (524288) all platforms except Windows Mobile

64 KB (65536 bytes) Windows Mobile

Remarks

The PrefetchBuffer (PBUF) connection parameter controls the per-connection maximum memory allocated on the client to store prefetched rows.

In some circumstances, increasing the number of prefetched rows can improve query performance. You can increase the number of prefetched rows using the PrefetchRows (PROWS) and PrefetchBuffer (PBUF) connection parameters.

Increasing the PrefetchBuffer (PBUF) connection parameter also increases the amount of memory used to buffer GET DATA requests. This may improve performance for some applications that process many GET DATA (SQLGetData) requests.

See also

- "Using connection parameters" on page 86

Example

The following connection string fragment could be used to determine if the PrefetchBuffer memory limit is reducing the number of prefetched rows.

```
...PrefetchRows=100;LogFile=c:\client.txt
```

The following string could be used to increase the memory limit to 2 MB:

```
...PrefetchRows=100;PrefetchBuffer=2M
```

PrefetchOnOpen connection parameter

Sends a prefetch request with a cursor open request when this parameter is enabled.

Syntax

PrefetchOnOpen={ **YES** | **NO** }

Usage

ODBC

Default

NO

Remarks

Enabling this option sends a prefetch request with a cursor open request, thereby eliminating a network request to fetch rows each time a cursor is opened. Columns must already be bound in order for the prefetch to occur on the open. Rebinding columns between the cursor open and the first fetch when using PrefetchOnOpen will cause reduced performance.

Making ODBC calls to SQLExecute or SQLExecDirect on a query or stored procedure which returns a result set causes a cursor open.

Enabling this option can improve performance if your:

- network exhibits poor latency

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- application sends many cursor open and close requests

PrefetchRows (PROWS) connection parameter

Provides an initial suggested number of rows to prefetch when querying the database.

Syntax

{ **PrefetchRows** | **PROWS** }=*number-of-rows*

Usage

Anywhere

Allowed values

- **number-of-rows** The number of rows prefetched is limited both by the PrefetchRows (PROWS) connection parameter and the PrefetchBuffer (PBUF) connection parameter, which limits the memory available for storing prefetched rows. See "PrefetchBuffer (PBUF) connection parameter" on page 303.

  The maximum number of rows that can be prefetched is 10000.

Default

10

200 for ADO.NET

Remarks

By default, the client library dynamically increases the number of prefetch rows for cursors that would gain a performance benefit from such an increase. Because of this behavior, increasing the PrefetchRow default does not significantly improve the performance of most applications.

Increasing the PrefetchRow default may improve the performance of ODBC, OLE DB, and SQL Anywhere JDBC applications that use STATIC or FORWARD ONLY FOR UPDATE cursor types but do mostly fetch next operations and very rare positioned updates or deletes, particularly if the client and database server are communicating over a slow or wide area network.

The number of prefetched rows is limited by the PrefetchBuffer (PBUF) connection parameter, which limits the memory available for storing prefetched rows. See "PrefetchBuffer (PBUF) connection parameter" on page 303.

The maximum number of rows that can be prefetched is 10000.

See also

- "Using connection parameters" on page 86

Example

The following connection string fragment sets the number of prefetched rows to 100:

```
...PrefetchRows=100;...
```

RetryConnectionTimeout (RetryConnTO) connection parameter

Instructs the client library (DBLIB, ODBC, ADO, and so on) to keep retrying the connection attempt, as long as the server is not found, for the specified period of time.

Syntax

{ **RetryConnectionTimeout** | **RetryConnTO** }=*timeout-value*

Usage

Anywhere

Allowed values

- **timeout-value** The value specified by this connection is a timeout, in seconds. It is not a counter of the number of times to retry the connection attempt. The default value of zero indicates that the connection attempt should only be tried once.

Default

0

Remarks

There is a half-second delay between iterations, and the retries only occur if the connection attempt failed because the database server was not found. Any other error is returned immediately. If the database server is not found, the connection attempt will take at least as long as the time specified by the RetryConnectionTimeout connection parameter.

Note that the default TCP timeout is 5 seconds, so if your connection string contains a value for RetryConnTO that is less than 5, for example `HOST=host-name;RetryConnTO=3`, then the connection attempt still takes 5 seconds.

See also

- "Timeout (TO) protocol option" on page 338

Example

The following connection string fragment tells the client library to continue to retry the connection attempt for at least 5 seconds:

```
...RetryConnTO=5;...
```

ServerName (Server) connection parameter

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Specifies the name of a running database server to which you want to connect.

Syntax

{ **ServerName** | **Server** }=*database-server-name-string*

Usage

Network servers or personal servers.

Allowed values

- **database-server-name-string** If you are starting a database server automatically, you can provide a server name using this parameter.

  The server name is interpreted according to the character set of the client computer. Non-ASCII characters are not recommended in server names.

  Names must be valid identifiers. Database server names cannot:

  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons
  - be longer than 250 bytes

  On Windows and Unix, version 9.0.2 and earlier clients cannot connect to version 10.0.0 and later database servers with names longer than the following lengths:

  - 40 bytes for Windows shared memory
  - 31 bytes for Unix shared memory
  - 40 bytes for TCP/IP

Default

The default local database server.

Remarks

EngineName and ENG are accepted for backwards compatibility, but are deprecated.

When a database server starts, it attempts to become the default database server on that computer. The first database server to start when there is no default server becomes the default database server. Shared memory connection attempts on that computer that do not explicitly specify a database server name connect to the default server.

ServerName is not needed if you want to connect to the default local database server.

If you are connecing over TCP/IP and use the HOST connection parameter, ServerName is not required if the host name is provided.

In the **Connect** window, and in the **ODBC Configuration For SQL Anywhere** window, this is the **Server Name** field.

Note
It is recommended that you include the ServerName parameter in connection strings for deployed applications. This ensures that the application connects to the correct server in the case where a computer is running multiple SQL Anywhere database servers, and can help prevent timing-dependent connection failures.

It is recommended that you use the -xd option for database servers being used by deployed applications, and that all clients explicitly specify the name of the database server to which they should connect by using the Server connection parameter. This ensures that the database connects to the correct database server when a computer is running multiple SQL Anywhere database servers.

See also

- "Identifiers" [*SQL Anywhere Server - SQL Reference*]
- "-n dbeng12/dbsrv12 server option" on page 206
- "-xd dbeng12/dbsrv12 server option" on page 239
- "Using connection parameters" on page 86
- "Connecting to an embedded database" on page 131

Example

Connect to a server named Guelph:

```
Server=Guelph
```

StartLine (START) connection parameter

Starts a local database server running from an application.

Syntax

{ **StartLine** | **START** }=*local-database-server-command*

Usage

Embedded databases

Allowed values

- **local-database-server-command** By default, SQL Anywhere attempts to connect to a running database server. If no server can be found with the specified connection parameters, SQL Anywhere automatically starts a new local database server using the *local-database-server-command*. The database server is not started automatically if the HOST connection parameter is specified or the CommLinks (LINKS) parameter includes TCP/IP. See "The SQL Anywhere database server" on page 147.

Default

dbeng12 on all platforms except for Windows Mobile.

dbsrv12 on Windows Mobile.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

The StartLine connection parameter is only used to start a database server if a connection cannot be made to the specified database server.

For example, suppose you start a database server running a database as follows:

```
dbeng12 c:\mydb.db
```

Connect another database (without specifying a database server name using the Server connection parameter):

```
dbisql -c "START=dbsrv12 -x none -c 8M;DBN=seconddb;DBF=c:
\myseconddb.db;UID=DBA;PWD=sql"
```

In this case, the dbsrv12 database server is not started. Instead, the dbeng12 database server that was used to start *mydb.db* is used to start and connect to *myseconddb.db*.

However, if Server=*server-name* had been specified, and a database server named *server-name* was not running, then the dbsrv12 database server would have started.

Note

If you want to specify the database name, database file, or server, it is recommended that you use the DBN, DBF, and Server connection parameters, rather than the StartLine connection parameter.

The following command uses the recommended syntax:

```
START=dbeng12 -c 8M;Server=mydb;DBN=mydb;DBF=c:\sample.db
```

The following syntax is not recommended:

```
START=dbeng12 -c 8M -n mydb "c:\sample.db"
```

See also

- "Using connection parameters" on page 86
- "CommLinks (LINKS) connection parameter" on page 272
- "Connecting to an embedded database" on page 131

Example

The following data source fragment starts a personal database server with a cache of 8 MB.

```
StartLine=dbeng12 -c 8M;DBF=samples-dir\demo.db
```

For information about *samples-dir*, see "Samples directory" on page 392.

Unconditional (UNC) connection parameter

Stops a database server using the db_stop_engine function, or a database using the db_stop_database function, even when there are connections to the database server.

Syntax

{ **Unconditional** | **UNC** }={ **YES** | **NO** }

Usage

db_stop_engine and db_stop_database functions only

Default

NO

Remarks

The db_stop_engine and db_stop_database functions shut down a database server or database, respectively. If you specify UNC=YES in the connection string, the database server or database is shut down even if there are active connections. If Unconditional is not set to YES, then the database server or database is shut down only if there are no active connections.

See also

- "db_stop_database function" [*SQL Anywhere Server - Programming*]
- "db_stop_engine function" [*SQL Anywhere Server - Programming*]
- "Using connection parameters" on page 86

Userid (UID) connection parameter

Specifies the user ID used to log in to the database.

Syntax

{ **Userid** | **UID** }=*userid*

Usage

Anywhere

Allowed values

- **userid** User IDs cannot:
  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons

Default

None

Remarks

You must always supply a user ID when connecting to a database, unless you are using an integrated or Kerberos login.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- “Using connection parameters” on page 86
- “Database permissions and authorities” on page 441

Example

The following connection string fragment supplies the user ID DBA and password sql:

```
UID=DBA;PWD=sql
```

Network protocol options

Network protocol options (for both the client and the server) enable you to work around peculiarities of different network protocol implementations.

You can supply the network protocol options in the server command. For example:

```
dbsrv12 -x tcpip(PARM1=value1;PARM2=value2;...)
```

From the client side, you enter the protocol options as the CommLinks (LINKS) connection parameter:

```
CommLinks=tcpip(PARM1=value1;PARM2=value2;...)
```

If there are spaces in a parameter, the network protocol options must be enclosed in quotation marks to be parsed properly by the system command interpreter:

```
dbsrv12 -x "tcpip(PARM1=value1;PARM2=value2;...)"
CommLinks="tcpip(PARM1=value1;PARM2=value2;...)"
```

The quotation marks are also required under Unix if more than one parameter is given because Unix interprets the semicolon as a command separator.

Boolean parameters are turned on with YES, Y, ON, TRUE, T, or 1, and are turned off with any of NO, N, OFF, FALSE, F, and 0. The parameters are case insensitive.

The examples provided should all be entered on a single line; you can also include them in a configuration file and use the *@data* server option to invoke the configuration file.

TCP/IP	HTTP	HTTPS	TLS
“Broadcast (BCAST) protocol option” on page 313	“DatabaseName (DBN) protocol option” on page 319	“certificate_company protocol option” on page 315	“certificate_company protocol option” on page 315
“BroadcastListener (BLISTENER) protocol option” on page 314	“KeepaliveTimeout (KTO) protocol option” on page 326	“certificate_name protocol option” on page 316	“certificate_name protocol option” on page 316

TCP/IP	HTTP	HTTPS	TLS
“ClientPort (CPORT) protocol option” on page 318	“LocalOnly (LOCAL) protocol option” on page 327	“certificate_unit protocol option” on page 317	“certificate_unit protocol option” on page 317
“DoBroadcast (DOBROAD) protocol option” on page 320	“LogFile (LOG) protocol option” on page 328	“DatabaseName (DBN) protocol option” on page 319	“FIPS protocol option” on page 325
“Host (IP) protocol option” on page 321	“LogFormat protocol (LF) option” on page 329	“Identity protocol option” on page 323	“LocalOnly (LOCAL) protocol option” on page 327
“LDAP protocol option” on page 326	“LogMaxSize (LSIZE) protocol option” on page 330	“Identity_Password protocol option” on page 324	“trusted_certificates protocol option” on page 339
“LocalOnly (LOCAL) protocol option” on page 327	“LogOptions (LOPT) protocol option” on page 331	“FIPS protocol option” on page 325	
“MyIP (ME) protocol option” on page 333	“MaxConnections (MAXCONN) protocol option” on page 332	“LocalOnly (LOCAL) protocol option” on page 327	
“ReceiveBufferSize (RCVBUFSZ) protocol option” on page 334	“MaxRequestSize (MAXSIZE) protocol option” on page 332	“LogFile (LOG) protocol option” on page 328	
“SendBufferSize (SNDBUFSZ) protocol option” on page 335	“MyIP (ME) protocol option” on page 333	“LogFormat protocol (LF) option” on page 329	
“ServerPort (PORT) protocol option” on page 335	“ServerPort (PORT) protocol option” on page 335	“LogMaxSize (LSIZE) protocol option” on page 330	
“TDS protocol option” on page 338	“Timeout (TO) protocol option” on page 338	“LogOptions (LOPT) protocol option” on page 331	
“Timeout (TO) protocol option” on page 338		“MaxConnections (MAXCONN) protocol option” on page 332	

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

TCP/IP	HTTP	HTTPS	TLS
“VerifyServerName (VERIFY) protocol option” on page 340		“MaxRequestSize (MAXSIZE) protocol option” on page 332	
		“MyIP (ME) protocol option” on page 333	
		“ServerPort (PORT) protocol option” on page 335	
		“Timeout (TO) protocol option” on page 338	
		“trusted_certificates protocol option” on page 339	

Broadcast (BCAST) protocol option

Specifies the IP address that should be used to send broadcast messages.

Syntax

{ **Broadcast** | **BCAST** }=*ip-address*

Available protocols

TCP/IP

Allowed values

- **ip-address** This string must be specified in the form of an IP address.

Default

Broadcasts to all addresses on the same subnet.

Remarks

The default broadcast address is created using the local IP address and subnet mask. The subnet mask indicates which portion of the IP address identifies the network, and which part identifies the host.

For example, for a subnet of 10.24.98.x, with a mask of 255.255.255.0, the default broadcast address would be 10.24.98.255.

When specifying an IPv6 address on a Windows platform, the interface identifier should be used. Unix platforms support both interface identifiers and interface names in IPv6 addresses. The interface identifier is required on Linux (kernel 2.6.13 and later). See "IPv6 support in SQL Anywhere" on page 79.

See also

- "BroadcastListener (BLISTENER) protocol option" on page 314
- "DoBroadcast (DOBROAD) protocol option" on page 320
- "Locating a database server using the Broadcast Repeater utility" on page 143

Example

The following connection string example tells the client to broadcast only on interface number 2 when using IPv6:

```
LINKS=tcpip(BROADCAST=ff02::1%2)
```

BroadcastListener (BLISTENER) protocol option

Controls broadcast listening for the specified port.

Syntax

{ **BroadcastListener** | **BLISTENER** }={ **YES** | **NO** }

Available protocols

TCP/IP (server side)

Default

YES

Remarks

This option allows you to turn broadcast listening off for this port.

Using **-sb 0** is the same as specifying `BroadcastListener=NO` on TCP/IP.

If broadcast listening is off, then the database server does not respond to UDP broadcasts. This means that clients must use either the HOST= TCP protocol option to specify the hostname of the database server, or register the database server with LDAP and use LDAP on the clients to find the database server. This also means that the dblocate utility does not include the database server in its output.

See also

- "-sb dbeng12/dbsrv12 server option" on page 219
- "Broadcast (BCAST) protocol option" on page 313
- "DoBroadcast (DOBROAD) protocol option" on page 320
- "Server Enumeration utility (dblocate)" on page 830
- "Host connection parameter" on page 291
- "Host (IP) protocol option" on page 321

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Example

Start a database server that accepts TCP/IP connections, and requires that TCP/IP connections use the Host connection parameter or Host protocol option:

```
dbsrv12 -x tcpip(BroadcastListener=NO) ...
```

The following is a fragment of a client connection string to connect to the database server:

```
...LINKS=tcpip(...HOST=myhost;...);...
```

certificate_company protocol option

Forces the client to accept server certificates only when the Organization field on the certificate matches this value.

Syntax

certificate_company= *organization*

Available protocols

TLS, HTTPS

Default

None

Remarks

SQL Anywhere clients trust all certificates signed by the certificate authority, so they may also trust certificates that the same certificate authority has issued to other companies. Without a means to discriminate, your clients might mistake a competitor's database server for your own and accidentally send it sensitive information. This option specifies a further level of verification, that the Organization field in the identity portion of the certificate also matches a value you specify.

HTTPS is only supported for web services client procedures. See "CREATE PROCEDURE statement (web clients)" [*SQL Anywhere Server - SQL Reference*].

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

See also

- “certificate_name protocol option” on page 316
- “certificate_unit protocol option” on page 317
- “trusted_certificates protocol option” on page 339
- “Encryption (ENC) connection parameter” on page 287
- “Encrypting SQL Anywhere client/server communications” on page 1152
- “Certificate Creation utility (createcert)” on page 775
- “-xs dbeng12/dbsrv12 server option” on page 241

Example

The following command connects the SQL Anywhere sample database to Interactive SQL using transport-layer security.

```
dbisql -c
"UID=DBA;PWD=sql;HOST=myhost;Server=demo;ENC=TLS(
tls_type=RSA;FIPS=n;trusted_certificates=c:\temp\myident;
certificate_unit='SA';certificate_company='Sybase iAnywhere';
certificate_name='Sybase')"
```

certificate_name protocol option

Forces the client to accept server certificates only when the Common Name field on the certificate matches this value.

Syntax

certificate_name=*common-name*

Available protocols

TLS, HTTPS

Default

None

Remarks

SQL Anywhere clients trust all certificates signed by the certificate authority, so they may also trust certificates that the same certificate authority has issued to other companies. Without a means to discriminate, your clients might mistake a competitor's database server for your own and accidentally send it sensitive information. This option specifies a further level of verification, that the Common Name field in the identity portion of the certificate also matches a value you specify.

HTTPS is only supported for web services client procedures. See “CREATE PROCEDURE statement (web clients)” [*SQL Anywhere Server - SQL Reference*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See “Separately licensed components” [*SQL Anywhere 12 - Introduction*].

See also

- “certificate_company protocol option” on page 315
- “certificate_unit protocol option” on page 317
- “trusted_certificates protocol option” on page 339
- “Encryption (ENC) connection parameter” on page 287
- “Encrypting SQL Anywhere client/server communications” on page 1152
- “Certificate Creation utility (createcert)” on page 775
- “-xs dbeng12/dbsrv12 server option” on page 241

Example

The following command connects the SQL Anywhere sample database to Interactive SQL using transport-layer security.

```
dbisql -c
"UID=DBA;PWD=sql;HOST=myhost;Server=demo;ENC=TLS(
tls_type=RSA;FIPS=n;trusted_certificates=c:\temp\myident;
certificate_unit='SA';certificate_company='Sybase iAnywhere';
certificate_name='Sybase')"
```

certificate_unit protocol option

Forces the client to accept server certificates only when the Organization Unit field on the certificate matches this value.

Syntax

certificate_unit=*organization-unit*

Available protocols

TLS, HTTPS

Default

None

Remarks

SQL Anywhere clients trust all certificates signed by the certificate authority, so they may also trust certificates that the same certificate authority has issued to other companies. Without a means to discriminate, your clients might mistake a competitor's database server for your own and accidentally send it sensitive information. This option specifies a further level of verification, that the Organization Unit field in the identity portion of the certificate also matches a value you specify.

HTTPS is only supported for web services client procedures. See "CREATE PROCEDURE statement (web clients)" [*SQL Anywhere Server - SQL Reference*].

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

See also

- "certificate_company protocol option" on page 315
- "certificate_name protocol option" on page 316
- "trusted_certificates protocol option" on page 339
- "Encryption (ENC) connection parameter" on page 287
- "Encrypting SQL Anywhere client/server communications" on page 1152
- "Certificate Creation utility (createcert)" on page 775
- "-xs dbeng12/dbsrv12 server option" on page 241

Example

The following command connects the SQL Anywhere sample database to Interactive SQL using transport-layer security.

```
dbisql -c
"UID=DBA;PWD=sql;HOST=myhost;Server=demo;ENC=TLS(
tls_type=RSA;FIPS=n;trusted_certificates=c:\temp\myident;
certificate_unit='SA';certificate_company='Sybase iAnywhere';
certificate_name='Sybase')"
```

ClientPort (CPORT) protocol option

Designates the port number on which the client application communicates using TCP/IP.

Syntax

{ **ClientPort** | **CPORT** }=*port-number*

Available protocols

TCP/IP (client side only)

Default

Assigned dynamically per connection by the networking implementation. If you do not have firewall restrictions, it is recommended that you do not use this parameter.

Remarks

This option is provided for connections across firewalls, as firewall software filters according to TCP/UDP port. It is recommended that you do not use this protocol option unless you need to for firewall reasons.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The ClientPort option designates the port number on which the client application communicates using TCP/IP. You can specify a single port number, or a combination of individual port numbers and ranges of port numbers. For example:

- (CPORT=1234)
- (CPORT=1234,1235,1239)
- (CPORT=1234-1238)
- (CPORT=1234-1237,1239,1242)

It is best to specify a list or a range of port numbers if you want to make multiple connections using a given Data Source or a given connect string. If you specify a single port number, then your application will be able to maintain only one connection at a time. In fact, even after closing the one connection, there is a several minute timeout period during which no new connection can be made using the specified port. When you specify a list and/or range of port numbers, the application keeps trying port numbers until it finds one to which it can successfully bind.

See also

- "Host (IP) protocol option" on page 321
- "DoBroadcast (DOBROAD) protocol option" on page 320
- "ServerPort (PORT) protocol option" on page 335
- "Connecting across a firewall" on page 80

Example

The following connection string fragment makes a connection from an application using port 6000 to a server named my-server using port 5000:

```
CommLinks=tcpip(ClientPort=6000;ServerPort=5000);ServerName=my-server
```

The following connection string fragment makes a connection from an application that can use ports 5050 through 5060, and ports 5040 and 5070 for communicating with a server named my-server using the default server port:

```
CommLinks=tcpip(ClientPort=5040,5050-5060,5070);
ServerName=my-server
```

DatabaseName (DBN) protocol option

Specifies the name of a database to use when processing web requests, or uses the REQUIRED or AUTO keyword to specify whether database names are required as part of the URL.

Syntax

{ **DatabaseName** | **DBN** }={ **AUTO** | **REQUIRED** | *database-name* }

Available protocols

HTTP, HTTPS

Default

AUTO

Remarks

If this parameter is set to REQUIRED, the URL must specify a database name.

If this parameter is set to AUTO, the URL may specify a database name, but does not need to do so. If the URL contains no database name, the default database on the server is used to process web requests. Since the server must determine whether the URL contains a database name when set to AUTO, you should avoid ambiguity in your web site design.

If this parameter is set to the name of a database, that database is used to process all web requests. The URL must not contain a database name.

Example

The following command starts two databases, but permits only one of them to be accessed via HTTP.

```
dbsrv12 -xs http( DBN=web ) samples-dir\demo.db web.db
```

The following command starts two HTTP web services—one for *your-first-database.db* and one for *your-second-database.db*:

```
dbsrv12 -xs
    http(port=80;dbn=your-first-database)
    http(port=8800;dbn=your-second-database)
    your-first-database.db your-second-database.db
```

DoBroadcast (DOBROAD) protocol option

Controls how a client searches for a database server, and controls whether the database server broadcasts when it starts.

Syntax (database server)

{ **DoBroadcast** | **DOBROAD** }= { **YES** | **NO** }

Syntax (client)

{ **DoBroadcast** | **DOBROAD** }= { **ALL** | **NONE** | **DIRECT** }

Available protocols

TCP/IP

Allowed values (database server)

- **YES** Setting DoBroadcast=YES allows the database server to broadcast to find other database servers with the same name when it starts.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **NO** Setting DoBroadcast=NO prevents the database server from broadcasting to find other database servers with the same name when starting up. This is useful in certain rare circumstances, but it is not generally recommended.

Allowed values (client)

- **ALL** With DoBroadcast=ALL a broadcast is performed to search for a database server. The broadcast goes first to the local subnet. If HOST= is specified, broadcast packets are also sent to each of the hosts. All broadcast packets are UDP packets.

- **NONE** Specifying DoBroadcast=NONE causes no UDP broadcasts to be used and the server address cache (*sasrv.ini*) is ignored. A TCP/IP connection is made directly with the HOST/PORT specified, and the server name is verified.

- **DIRECT** With DoBroadcast=DIRECT, no broadcast is performed to the local subnet to search for a database server. Broadcast packets are sent only to the hosts listed in the HOST (IP) protocol option. If you specify DoBroadcast=DIRECT, the HOST (IP) protocol option is required.

Default

YES (database server)

ALL (client)

Remarks

Client usage With TCP/IP, you can choose not to verify the server name by setting the VerifyServerName (VERIFY) protocol option to NO. The HOST (IP) protocol option is a required parameter, unless LDAP is being used, while the ServerPort (PORT) protocol option is optional.

For DIRECT and NONE, you must specify the database server host with the HOST option.

See also

- “Broadcast (BCAST) protocol option” on page 313
- “BroadcastListener (BLISTENER) protocol option” on page 314

Example

The following command starts a client without broadcasting to search for a database server. Instead, the server is looked for only on the computer named silver.

```
CommLinks=tcpip(DOBROADCAST=DIRECT;HOST=silver) demo
```

Host (IP) protocol option

Specifies additional computers outside the immediate network to be searched by the client library.

Syntax

{ **Host** | **IP** }=*ip-address*

Available protocols

TCP/IP

Allowed values

- **ip-address** This string must be specified in the form of an IP address and it can optionally include a port number (separated by a colon). The list of host values is a comma-separated list. You can use **localhost** to identify the current computer. For example:

```
links=tcpip(HOST=myhost)
links=tcpip(HOST=myhost:1234)
links=tcpip(HOST=10.25.13.5,myotherhost)
links=tcpip(HOST=myhost:1234,10.25.65.112)
links=tcpip(HOST=myhost:1234,myotherhost:5678)
```

Default

No additional computers.

Remarks

HOST specifies additional computers outside the immediate network to be searched by the client library. For TCP/IP, the address can be the *hostname* IP address. You may optionally specify a PORT value as well. If a port is specified, only that port number is used for TCP/IP connections and UDP broadcasts. If a port number is not specified, port 2638 is used.

By default, the client does not broadcast to find the server. You can change this behavior by setting the DoBroadcast protocol option to Direct. See "DoBroadcast (DOBROAD) protocol option" on page 320.

When specifying an IPv6 address on a Windows platform, the interface identifier should be used. Unix platforms support both interface identifiers and interface names in IPv6 addresses. The interface identifier is required on Linux (kernel 2.6.13 and later). See "IPv6 support in SQL Anywhere" on page 79.

The server prints addressing information to the database server messages window during startup if the -z option is used. See "-z dbeng12/dbsrv12 server option" on page 243.

In addition, the client application writes this information to its log file if the LogFile connection parameter is specified. See "LogFile (LOG) connection parameter" on page 298.

You can use a comma-separated list of addresses to search for more than one computer. You can also append a port number to an IP address, using a colon as separator. Alternatively, you can specify the host and server ports explicitly, as in `HOST=myhost;PORT=5000`. For IPv6 addresses, you must enclose the address in parentheses, for example `(fe80::5445:5245:444f):2638`.

To specify multiple values for a single parameter, use a comma-separated list. IP and HOST are synonyms.

See also

- "ClientPort (CPORT) protocol option" on page 318

Example

The following connection string fragment instructs the client to look on the computers kangaroo and 197.75.209.222 (port 2369) to find a database server:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
LINKS=tcpip(IP=kangaroo,197.75.209.222:2369)
```

The following connection string fragment instructs the client to look on the computers my-server and kangaroo to find a database server. A connection is attempted to the first host that responds running on port 2639.

```
LINKS=tcpip(HOST=my-server,kangaroo;PORT=2639)
```

The following connection string fragment instructs the client to look for a server on host1 running on port 1234 and for a server on host2 running on port 4567. The client does not look on host1 on port 4567 or on host2 on port 1234.

```
LINKS=tcpip(HOST=host1:1234,host2:4567)
```

The following connection string fragment instructs the client to look for a server on an IPv6 address:

```
LINKS=tcpip(HOST=fe80::5445:5245:444f)
```

The following examples demonstrate using IPv6 addresses with the Host protocol option:

```
Global scope address, unique everywhere, so no interface index is required
// no index required
-c "links=tcpip(Host=fd77:55d:59d9:56a:202:55ff:fe76:df19)"
// all communication is done through interface 2
-c "links=tcpip(Host=fd77:55d:59d9:56a:202:55ff:fe76:df19%2)"
// all communication is done through eth0
-c "links=tcpip(Host=fd77:55d:59d9:56a:202:55ff:fe76:df19%eth0)"

Link scope address, addresses are unique on each interface
// possibly ambiguous (this host may exist through both eth0 and eth1)
-c "links=tcpip(Host=fe80::202:55ff:fe76:df19)"
// not ambiguous because it must use interface 2
-c "links=tcpip(Host=fe80::202:55ff:fe76:df19%2)"
// not ambiguous because it must use eth0
-c "links=tcpip(Host=fe80::202:55ff:fe76:df19%eth0)"
```

Identity protocol option

Specifies the name of an identity file.

Syntax

Identity=*identity-file*

Available protocols

HTTPS

Allowed values

- **identity-file** This string specifies the name of an identity file.

Default

There is no default identity file name.

Remarks

When you use transport-layer security, the identity file contains the public certificate and its private key, and for certificates that are not self-signed, the identity file also contains all the signing certificates, which includes, among other things, the encryption certificate. The password for this certificate must be specified with the Identity_Password parameter.

See also

- "Setting up transport-layer security" on page 1145
- "Identity_Password protocol option" on page 324

Example

Start a server that requires web connections to use a particular encryption certificate.

```
dbsrv12 -xs https(Identity=cert.file;Identity_Password=secret) ...
```

Caution
The sample identity file is intended for use only during testing and development. It provides no protection because it is a standard part of SQL Anywhere. Replace it with your own certificate before deploying your application.

Identity_Password protocol option

Specifies the password for the encryption certificate.

Syntax

Identity_Password=*password*

Available protocols

HTTPS

Allowed values

- **password** This string specifies the password for an encryption certificate.

Default

There is no default identity file password.

Remarks

When you use transport-layer security, this option specifies the password that matches the password for the encryption certificate specified by the Identity protocol option.

See also

- "Setting up transport-layer security" on page 1145
- "Identity protocol option" on page 323

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Example

Start a server that requires web connections to use a particular encryption certificate.

```
dbsrv12 -xs https(Identity=cert.file;Identity_Password=secret) ...
```

FIPS protocol option

Allows you to use of FIPS-approved security algorithms to encrypt database files, communications for database client/server communication, and web services.

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

Syntax

{ **FIPS** }={ **YES** | **NO** }

Available protocols

TLS, HTTPS

Allowed values

YES, NO

Default

NO

Remarks

FIPS is only supported for RSA encryption. Non-FIPS clients can connect to FIPS servers and vice versa. This option can be used with end-to-end encryption.

If fips is set to YES, MobiLink clients use FIPS 140-2 certified implementations of RSA and AES. This option is not supported when using ECC.

See also

- "certificate_company protocol option" on page 315
- "certificate_name protocol option" on page 316
- "certificate_unit protocol option" on page 317
- "trusted_certificates protocol option" on page 339
- "Encryption (ENC) connection parameter" on page 287
- "Encrypting SQL Anywhere client/server communications" on page 1152
- "Certificate Creation utility (createcert)" on page 775
- "-ec dbeng12/dbsrv12 server option" on page 176
- "-xs dbeng12/dbsrv12 server option" on page 241

Example

The following command line configures the PORT, FIPS, Identity, and Identity_Password network protocol options for a web server:

```
dbsrv12 -xs https(PORT=544;FIPS=YES;
    Identity=certificate.id;Identity_Password=password) your-database-name.db
```

KeepaliveTimeout (KTO) protocol option

Specifies the maximum time, in seconds, that the database server waits for a complete request.

Syntax

{ **KeepaliveTimeout** | **KTO** }=*timeout-value*

Available protocols

HTTP

Allowed values

- **timeout-value** This integer specifies the length of time, in seconds, to wait for a response when establishing communications. If you do not want the connection to time out, specify KTO=0.

Default

60

Remarks

Normally, a connection is closed after each request. When a client requests the keep-alive option, an HTTP connection can be kept open after each request and response, so that multiple requests can be executed on the same connection.

Once a connection is opened, the client has the specified amount of time to send the complete HTTP request, including the body for POST requests. On connections where keep-alive is requested, the timeout is reset after sending a result, so the beginning of each request is like the opening of a new connection.

The difference between the KeepaliveTimeout and Timeout protocol options is that KeepaliveTimeout specifies the total time from opening the connection, while Timeout specifies the maximum amount of time between packets within the request.

See also

- "Timeout (TO) protocol option" on page 338

LDAP protocol option

Allows clients to find database servers without specifying the IP address.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

LDAP={ **YES** | **NO** | *filename* }

Usage

TCP/IP

Allowed values

- **YES** Specifying LDAP=YES turns LDAP support on and uses *saldap.ini* (the default file name) as the configuration file.
- **NO** Specifying NO turns LDAP support off.
- **filename** Specifying LDAP=*filename* turns LDAP support on and uses the specified file as the configuration file.

Default

YES

Remarks

Having the database server register itself with an LDAP server allows clients to query the LDAP server. This allows clients running over a WAN or through a firewall to find servers without specifying the IP address. It also allows the Locate utility (dblocate) to find such servers.

You can hide the contents of the *saldap.ini* file with simple encryption using the File Hiding utility. See "Hiding the contents of .ini files" on page 397.

LDAP is only used with TCP/IP.

See also

- "Connecting using an LDAP server" on page 81

LocalOnly (LOCAL) protocol option

Allows a client to choose to connect only to a server on the local computer, if one exists.

Syntax

{ **LocalOnly** | **LOCAL** }={ **YES** | **NO** }

Available protocols

TCP/IP, HTTP, HTTPS

Default

NO

Remarks

If no database server with the matching server name is found on the local computer, a database server is not started automatically.

The LocalOnly (LOCAL) protocol option is only useful if DoBroadcast=ALL (the default) is also specified.

LocalOnly=YES uses the regular broadcast mechanism, except that broadcast responses from servers on other computers are ignored.

You can use the LocalOnly (LOCAL) protocol option with the server to restrict connections to the local computer. Connection attempts from remote computers will not find this server, and the Locate (dblocate) utility will not see this server. Running a server with the LocalOnly (LOCAL) protocol option set to YES allows the network server to run as a personal server without experiencing connection or CPU limits.

When set to YES, this parameter causes a network database server to reject all connections from clients running on different computers. This option has no effect on personal database servers, which never accept web service requests from other computers. The default value is NO, which means accept requests from clients no matter where they are located.

See also

- "Broadcast (BCAST) protocol option" on page 313
- "Starting an HTTP web server" [*SQL Anywhere Server - Programming*]

LogFile (LOG) protocol option

Specifies the name of the file where the database server writes information about web requests.

Syntax

{ **LogFile** | **LOG** }=*filename*

Available protocols

HTTP, HTTPS

Default

None

Remarks

This protocol option specifies the name of the file to which the database server writes information about web requests.

See also

- "LogFormat protocol (LF) option" on page 329
- "LogMaxSize (LSIZE) protocol option" on page 330
- "LogOptions (LOPT) protocol option" on page 331

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

LogFormat protocol (LF) option

Controls the format of messages written to the log file where the database server writes information about web requests, and specifies which fields appear in the messages.

Syntax

{ **LogFormat** | **LF** }=*format-string*

Available protocols

HTTP, HTTPS

Allowed values

- **format-string** The following codes are supported:
  - **@@** The @ character.
  - **@B** Date and time that processing of the request started, unless the request could not be queued due to an error.
  - **@C** Date and time that the client connected.
  - **@D** Name of the database associated with the request.
  - **@E** Text of the error message, if an error occurred.
  - **@F** Date and time that processing of the request finished.
  - **@I** IP address of the client.
  - **@J** Log the client port specified by the @I option.
  - **@L** Length of the response, in bytes, including headers and body.
  - **@M** HTTP request method.
  - **@P** Listener port associated with the request.
  - **@Q** Date and time that the request was queued for processing, unless the request could not be queued due to an error.
  - **@R** Status code and description of the HTTP response.
  - **@S** HTTP status code.
  - **@T** Date and time that the current log entry was written.
  - **@U** Requested URI.
  - **@V** Requested HTTP version.

- **@W** Time taken to process the request (@F - @B), or 0.000 if the request was not processed due to an error.

Default

@T - @W - @I:@J - @P - "@M @U @V" - @R - @L - @E

Remarks

This protocol option controls the format of messages written to the log file that stores information about web requests and which fields appear in them. If they appear in the string, the current values are substituted for the codes as each message is written.

See also

- "LogFile (LOG) protocol option" on page 328
- "LogMaxSize (LSIZE) protocol option" on page 330
- "LogOptions (LOPT) protocol option" on page 331

LogMaxSize (LSIZE) protocol option

Controls the maximum size of the log file where the database server writes information about web requests.

Syntax

{ **LogMaxSize** | **LSIZE** }=*size*[**k** | **m** | **g**]

Available protocols

HTTP, HTTPS

Allowed values

- **size** This integer specifies the maximum size of the file where web request information is written. The default value is in bytes, but you can use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively. If LogMaxSize is zero, the log file size is unlimited.

Default

0

Remarks

When the log file reaches the stated size, it is renamed and another log file is created.

See also

- "LogFile (LOG) protocol option" on page 328
- "LogFormat protocol (LF) option" on page 329
- "LogOptions (LOPT) protocol option" on page 331

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

LogOptions (LOPT) protocol option

Specifies the types of messages that are recorded in the log where the database server writes information about web requests.

Syntax

{ **LogOptions** | **LOPT** }= [**NONE**] [, **OK**] [, **INFO**] [, **ERRORS**] [, **ALL**] [, *status-codes*] [, **REQHDRS**] [, **RESHDRS**] [, **HEADERS**]

Available protocols

HTTP, HTTPS

Allowed values

The following keywords control which categories of messages are logged:

- **NONE** Log nothing.
- **OK** Log requests that complete successfully (20x HTTP status codes).
- **INFO** Log requests that return over or not modified status codes (3xx HTTP status codes).
- **ERRORS** Log all errors (4xx and 5xx HTTP status codes).
- **USER** Log all errors (6xx, 7xx, 8xx, and 9xx HTTP status codes).
- **ALL** Log all requests.

The following common HTTP status codes are also available. They can be used to log requests that return particular status codes:

- **C200** OK
- **C400** Bad request
- **C401** Unauthorized
- **C403** Forbidden
- **C404** Not found
- **C408** Request timeout
- **C501** Not implemented
- **C503** Service unavailable

The following keywords can be used to obtain more information about the logged messages:

- **REQHDRS** When logging requests, also write request headers to the log file.

- **RESHDRS** When logging requests, also write response headers to the log file.
- **HEADERS** When logging requests, also write both request and response headers to the log file (same as REQHDRS,RESHDRS).

Default

ALL

Remarks

The values available include keywords that select particular types of messages, and HTTP status codes. Multiple values may be specified, separated by commas.

See also

- "LogFile (LOG) protocol option" on page 328
- "LogFormat protocol (LF) option" on page 329
- "LogMaxSize (LSIZE) protocol option" on page 330

MaxConnections (MAXCONN) protocol option

Specifies the number of simultaneous connections accepted by the database server.

Syntax

{ **MaxConnections** | **MAXCONN** }=*size*

Available protocols

HTTP, HTTPS

Allowed values

- **size** This integer specifies the number of simultaneous connections accepted by the server. The value 0 indicates no limit.

Default

5 (personal server)

Number of licensed connections (network server)

See also

- "MaxRequestSize (MAXSIZE) protocol option" on page 332
- "Server Licensing utility (dblic)" on page 833

MaxRequestSize (MAXSIZE) protocol option

Specifies the size of the largest request the database server can accept.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

{ **MaxRequestSize** | **MAXSIZE** }=*size*[**k** | **m** | **g**]

Available protocols

HTTP, HTTPS

Allowed values

- **size** This integer specifies the size of the largest request the database server can accept. The default value is in bytes, but you can use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively. The value 0 disables this limit, but should be used with extreme caution. Without this limit, a rogue client could overload the database server or cause it to run out of memory.

Default

100k

Remarks

If the size of a request exceeds this limit, the connection is closed and the client returns a response indicating that the request is too large. This value limits only the size of the request, not that of the response.

See also

- "MaxConnections (MAXCONN) protocol option" on page 332

Example

The following command instructs the database server to accept requests up to 150000 bytes in size:

```
dbsrv12 -xs http{MaxRequestSize=150000}
```

MyIP (ME) protocol option

Indicates the networking interfaces that the client should use to find the database server when used on the client. Indicates the networking interfaces on which the database server should listen for connections when used on the database server.

Syntax

{ **MyIP** | **ME** }={ *ip-address* [**,** *ip-address* ...] | **NONE** }

Available protocols

TCP/IP, HTTP, HTTPS

Allowed values

- **ip-address** This string must be specified in the form of an IP address. You must separate multiple IP addresses with commas.

- **NONE** If the keyword NONE is supplied as the IP number, no attempt is made to determine the addressing information. The NONE keyword is intended for clients on computers where this operation

is expensive, such as computers with multiple network cards or remote access (RAS) software and a network card. It is not intended for use on the database server.

Remarks

The MyIP (ME) protocol option is provided for computers with more than one network adapter.

Each adapter has an IP address. By default, the database server uses every network interface it finds. If you don't want your database server to listen on all network interfaces, specify the address of each interface you want to use in the MyIP (ME) protocol option.

When specifying an IPv6 address on a Windows platform, the interface identifier should be used. Unix platforms support both interface identifiers and interface names in IPv6 addresses. The interface identifier is required on Linux (kernel 2.6.13 and later). See "IPv6 support in SQL Anywhere" on page 79.

See also

- "Using the TCP/IP protocol" on page 78

Example

The following command line (entered all on one line) instructs the server to use two network cards.

```
dbsrv12 -x tcpip(MyIP=192.75.209.12,192.75.209.32) "samples-dir\demo.db"
```

The following command line (entered all on one line) instructs the database server to use an IPv6 network card:

```
dbsrv12 -x tcpip(MyIP=fe80::5445:5245:444f) "samples-dir\demo.db"
```

For information about *samples-dir*, see "Samples directory" on page 392.

The following connection string fragment instructs the client to make no attempt to determine addressing information.

```
LINKS=tcpip(MyIP=NONE)
```

ReceiveBufferSize (RCVBUFSZ) protocol option

Sets the size for a buffer used by the TCP/IP protocol stack.

Syntax

{ **ReceiveBufferSize** | **RCVBUFSZ** }=*size*[**k** | **m** | **g**]

Available protocols

TCP/IP

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Allowed values

- **size** This integer specifies the size of the largest request the database server can accept. The default value is in bytes, but you can use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively.

Default

Computer-dependent.

Remarks

You may want to increase the value if BLOB performance over the network is important.

See also

- "Using the TCP/IP protocol" on page 78

SendBufferSize (SNDBUFSZ) protocol option

Sets the size for a buffer used by the TCP/IP protocol stack.

Syntax

{ **SendBufferSize** | **SNDBUFSZ** }=*size*[**k** | **m** | **g**]

Available protocols

TCP/IP

Allowed values

- **size** This integer specifies the size of the largest request the database server can accept. The default value is in bytes, but you can use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively.

Default

Computer-dependent.

Remarks

You may want to increase the value if BLOB performance over the network is important.

See also

- "Using the TCP/IP protocol" on page 78

ServerPort (PORT) protocol option

Specifies the port the database server is running on.

Syntax

{ **ServerPort** | **PORT** }=*port-number*

Available protocols

TCP/IP, HTTP, HTTPS

Allowed values

- **port-number** For a database server, the ServerPort protocol option designates the port number on which to communicate using TCP/IP.

  You can specify a single port number, or a combination of individual port numbers and ranges of port numbers. When you specify a list and/or range of port numbers, the database server attempts to bind to all specified port numbers. For example:

  - (port=1234)
  - (port=1234,1235,1239)
  - (port=1234-1238)
  - (port=1234-1237,1239,1242)

  For a client, the ServerPort (PORT) protocol option informs the client of the port or ports on which database servers are listening for TCP/IP communication. The client broadcasts to every port that is specified by the ServerPort (PORT) protocol option to find the database server.

Default

- **TCP/IP** 2638
- **HTTP** 80
- **HTTPS** 443

Remarks

The Internet Assigned Numbers Authority has assigned the SQL Anywhere database server port number 2638 to use for TCP/IP communications. However, other applications are not disallowed from using this reserved port, and this may result in an addressing collision between the database server and another application.

When you specify the ServerPort (PORT) protocol option, only the specified port number is used for TCP/IP connections and UDP broadcasts. If you do not specify the ServerPort protocol option, port 2638 is used. Applications can connect to the database server without specifying a port number. Having port 2638 available allows SQL Anywhere clients to find SQL Anywhere database servers running on other subnets and through firewalls.

Then the server listens to the same UDP port as the TCP/IP port. The database server listens to UDP ports and responds to requests on these ports so that clients can locate the database server by server name.

If you using a web server, by default, the database server listens on the standard HTTP and HTTPS ports of 80 and 443, respectively.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

UDP packets sent by the database server in response to client broadcasts contain no sensitive information. The data contained in these packets is limited to:

- database server name
- port number
- database server version
- names of databases running on the database server

You can hide database names from broadcast requests by using the -dh option. You can also specify -sb 0 to disable the UDP listeners completely.

Differences on Mac OS X

Mac OS X does not allow multiple processes to bind to the same UDP port. When the database server is running on one of these platforms, it only listens to the specified UDP port or port 2638 if no port is specified.

This means that clients must specify the TCP/IP port number if the database server is not using the default port (2638).

For example if the database server is started with the command `dbsrv12 -n MyServer samples-dir/demo.db`, a client on the same subnet can find the server using the following connection parameters `Server=MyServer;LINKS=tcpip`. If another server is started on Mac OS X, with the following command `dbsrv12 -n SecondServer -x tcpip(PORT=7777) samples-dir/demo.db`, a client on the same subnet can find the server using the connection parameters `Server=SecondServer;LINKS=tcpip(PORT=7777)`. Note that if the database server was running on a platform other than Mac OS X, then the client would not need to specify the PORT parameter.

Additionally, on Mac OS X, if a SQL Anywhere database server is already using port 2638, and a second network database server was started without the PORT protocol option, the second network server would fail to start. The reason for this is users need to know and specify the server's port number in their connection parameters. Personal servers start successfully, even if port 2638 is in use, because shared memory is normally used to connect to personal servers.

See also

- “-x dbeng12/dbsrv12 server option” on page 236
- “-xs dbeng12/dbsrv12 server option” on page 241
- “-sb dbeng12/dbsrv12 server option” on page 219

Example

The following example shows how to use the PORT protocol option to specify the port the database server starts on.

1. Start a network database server:

   ```
   dbsrv12 -x tcpip -n server1
   ```

   Port number 2638 is now taken.

2. Attempt to start another database server:

```
dbsrv12 -x tcpip -n server2
```

The default port is currently allocated, and so the server starts on another port. On Mac OS X, this will fail.

3. If another web server on your computer is already using port 80 or you do not have permission to start a server on this low of a port number, you may want to start a server that listens on an alternate port, such as 8080:

```
dbsrv12 -xs http(port=8080) -n server3 web.db
```

TDS protocol option

Controls whether TDS connections to a database server are allowed.

Syntax

TDS={ **YES** | **NO** }

Available protocols

TCP/IP (server side only)

Default

YES

Remarks

To disallow TDS connections to a database server, set TDS to NO. If you want to ensure that only encrypted connections are made to your database server, this protocol option is the only way to disallow TDS connections.

See also

- "-ec dbeng12/dbsrv12 server option" on page 176

Example

The following command starts a database server using the TCP/IP protocol, but disallowing connections from Sybase Open Client or jConnect applications.

```
dbsrv12 -x tcpip( TDS=NO ) ...
```

Timeout (TO) protocol option

Specifies the length of time, in seconds, to wait for a response when establishing communications.

Syntax

{ **Timeout** | **TO** }=*timeout-value*

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Available protocols

TCP/IP, HTTP, HTTPS

Allowed values

- **timeout-value** This integer specifies the length of time, in seconds, to wait for a response when establishing communications. The value 0 disables idle timeout, but should be used with extreme caution. Without this limit, a rogue client could consume the server's resources and prevent other clients from connecting.

Default

- **TCP/IP** 5
- **HTTP** 30
- **HTTPS** 30

Remarks

Timeout also specifies the length of time to wait for a response when disconnecting. You may want to try longer times if you are having trouble establishing TCP/IP communications.

On the database server, this is the amount of time to wait after sending the broadcast looking for servers with the same name. It is only used on server startup, and does not affect client connections.

When using HTTP or HTTPS on the server, this parameter specifies the maximum idle time permitted when receiving a request. If this limit is reached, the connection is closed and a request timeout message is returned to the client.

See also

- "KeepaliveTimeout (KTO) protocol option" on page 326

Example

The following data source fragment starts a TCP/IP communication link only, with a timeout period of twenty seconds.

```
...
CommLinks=tcpip(TO=20)
...
```

trusted_certificates protocol option

Specifies the path and file name of a file that contains one or more trusted certificates.

Syntax

trusted_certificates=*public-certificate*

Available protocols

TLS, HTTPS

Default

None

Remarks

Clients use the trusted_certificates encryption protocol option to specify trusted database server certificates. The trusted certificate can be a server's self-signed certificate, a public enterprise root certificate, or a certificate belonging to a commercial Certificate Authority. If you are using FIPS-approved RSA encryption, you must generate your certificates using RSA. If TLS is specified in the Encryption connection parameter, this protocol option is required.

HTTPS is only supported for web services client procedures. See "CREATE PROCEDURE statement (web clients)" [*SQL Anywhere Server - SQL Reference*].

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

See also

- "certificate_company protocol option" on page 315
- "certificate_name protocol option" on page 316
- "certificate_unit protocol option" on page 317
- "Encryption (ENC) connection parameter" on page 287
- "Encrypting SQL Anywhere client/server communications" on page 1152
- "Certificate Creation utility (createcert)" on page 775
- "-xs dbeng12/dbsrv12 server option" on page 241

Example

The following command connects the SQL Anywhere sample database to Interactive SQL using transport-layer security.

```
dbisql -c
"UID=DBA;PWD=sql;HOST=myhost;Server=demo;ENC=TLS(
tls_type=RSA;FIPS=n;trusted_certificates=c:\temp\myident;
certificate_unit='SA';certificate_company='Sybase iAnywhere';
certificate_name='Sybase')"
```

VerifyServerName (VERIFY) protocol option

Controls whether clients must verify the database server name before connecting.

Syntax

{ **VerifyServerName** | **VERIFY** }={ **YES** | **NO** }

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Available protocols

TCP/IP (client side only)

Default

YES

Remarks

When connecting over TCP using the DoBroadcast=NONE parameter, the client makes a TCP connection, then verifies that the name of the server found is the same as the one it is looking for. Specifying VerifyServerName=NO skips the verification of the server name. This allows SQL Anywhere clients to connect to a SQL Anywhere server if they know only an IP address/port.

The server name must still be specified in the connection string, but it is ignored. The VerifyServerName (VERIFY) protocol option is used only if DoBroadcast=NONE is specified.

If the server is using -sb 0 or BroadcastListener=NO, the client does not need to specify DoBroadcast=NONE to connect to it, although the client must still specify HOST=. The dblocate utility will not find the server.

Note
It is recommended that you only use this parameter in the rare circumstance where it is not possible to give each server a unique server name, and use that unique name to connect. Giving each server a unique server name, and connecting to the server using that name is still the best way to connect.

See also

- "DoBroadcast (DOBROAD) protocol option" on page 320

SQL Anywhere for Windows Mobile

SQL Anywhere allows the deployment of both client and server applications to the Windows Mobile device. This deployment model allows the database application to run autonomously. SQL Anywhere provides a network database server that runs on the Windows Mobile device. The network database server supports communications over TCP/IP. Because Windows Mobile supports the network database server, you can run administration utilities on a desktop computer to execute tasks on your Windows Mobile database. For example:

- You can use Sybase Central on your desktop computer to manage your database.
- You can use Interactive SQL on your desktop computer to load and unload data, and perform queries.

For more information, see "Using the administration utilities on Windows Mobile" on page 363.

See also

- "Tutorial: Running Windows Mobile databases from Sybase Central" on page 363

Installing SQL Anywhere on a Windows Mobile device

Requirements

- A computer running a supported Windows operating system
  - For Windows Vista and later: Windows Mobile Device Center 6.1 or later
  - For Windows XP or earlier: Microsoft ActiveSync 3.5 or later
- A Windows Mobile device supported by SQL Anywhere

  For a list of Windows Mobile devices supported by SQL Anywhere, see http://www.sybase.com/detail?id=1002288.

Windows Mobile file locations

The location of your SQL Anywhere install on Windows Mobile depends on the type of device and location you are installing to. No subdirectories are created. All DLLs are installed in the *\Windows* directory.

Location	Installation directory
Storage card	*\storage-card\Sybase SQL Anywhere 12*
Storage card	*\storage-card\SQLAny12*
Main memory	*\Program Files\SQLAny12*

Caution
It is recommended that you do not install SQL Anywhere on a storage card, such as an SD card.

When the Windows Mobile device resumes from being suspended, all open files, including executables and DLLs, located on a removable device may be closed by the operating system. The operating system itself loses access to the executables and DLLs that were in use by programs that were running when the device was suspended. In this case, the operating system may silently remove the process (such as the SQL Anywhere database server) from the system without an error message.

Installation considerations: Using ICU on Windows Mobile

The Unicode Collation Algorithm (UCA) is an algorithm for sorting the entire Unicode character set. It provides linguistically correct comparison, ordering, and case conversion. The UCA was developed as part of the Unicode standard. SQL Anywhere implements the UCA using the International Components for Unicode (ICU) open source library, developed and maintained by IBM.

On Windows Mobile, you require ICU if UCA is used as the NCHAR collation or the CHAR collation. You also require ICU on Windows Mobile if your CHAR character set does not match your operating system character set.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

By default, the ICU library is not installed on Windows Mobile because it adds approximately 1.7 MB to the size of the SQL Anywhere installation on Windows Mobile. However, you can modify your SQL Anywhere installation if you require the ICU library.

If you do not install the ICU library, you must choose either a collation whose character set matches the Windows Mobile character set or the UTF8BIN collation as the CHAR collation when creating your database. You must choose the UTF8BIN collation as the NCHAR collation when creating your database.

Creating databases on the desktop to deploy to Windows Mobile
When creating a database on the desktop to deploy to a Windows Mobile device, you can only use the UCA collation if the ICU library is installed on the Windows Mobile device. A database that uses the UCA is unusable on Windows Mobile if the ICU library is not installed on the device.

For more information about ICU, see "Unicode Collation Algorithm (UCA)" on page 420 and "Character set conversion" on page 410.

Installation considerations: Using the .NET Compact Framework on Windows Mobile

Although ADO.NET 3.5 is the most recent version of the API, the majority of devices that SQL Anywhere supports have only ADO.NET 1.x support installed. To use ADO.NET version 2.0 or 3.5 on a device, download and install the support for ADO.NET 2.0 or 3.5 from Microsoft on your device.

- **Version 2.0** For information about developing an application with ADO.NET 2.0, see "SQL Anywhere .NET Data Provider" [*SQL Anywhere Server - Programming*] and "Namespace" [*SQL Anywhere Server - Programming*].

- **Version 3.5** For information about developing an application with ADO.NET 3.5, see "SQL Anywhere .NET Data Provider" [*SQL Anywhere Server - Programming*] and "Namespace" [*SQL Anywhere Server - Programming*].

For more information about using ADO.NET, see "Tutorial: Using the SQL Anywhere .NET Data Provider" [*SQL Anywhere Server - Programming*].

Installation considerations: Limitations on Windows Mobile 5.0 for smartphone

All SQL Anywhere Server Windows Mobile functionality is supported on the smartphone, with the following limitations for Windows Mobile 5.0:

- **The shared memory protocol is not supported** TCP/IP is used, even if you do not specify a communication protocol. You must always specify a database server name when making a connection; if you do not, the connection fails.

- **The Server Startup Options window is not supported** The **Server Startup Options** window appears when you start the database server and do not specify any options. If you provide an incomplete or incorrect command when trying start the database server, an error appears and the database server does not start.

- **ODBC connections may not prompt for connection information** When you use the ODBC DriverCompletion parameter to SQLDriverConnect, you may be prompted for additional connection information. This prompt does not appear. If SQLDriverConnect fails, it does not prompt, and it returns an error.

- **Unload/reload is not supported** You must rebuild the Windows Mobile database on another platform and then copy the database to the Windows Mobile device. This is the recommended method for rebuilding a Windows Mobile database. See:

  - "Rebuilding databases on Windows Mobile" on page 359
  - "Rebuilding databases" [*SQL Anywhere Server - SQL Usage*]

Install SQL Anywhere for Windows Mobile

Use the following procedure to install SQL Anywhere for Windows Mobile on your Windows Mobile device.

To install SQL Anywhere for Windows Mobile

1. Connect your Windows Mobile device to a desktop computer running a supported Windows operating system.

2. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Deploy to Windows Mobile**.

3. Follow the instructions in the **SQL Anywhere for Windows Mobile Deployment Wizard**.

Using the Windows Mobile sample applications

The sample database that is used by the sample applications is named *demo.db* and is located in the *\My Documents* directory on Windows Mobile devices. Unlike the version of the sample database that is provided with the Windows version of SQL Anywhere, this one does not require the International Components for Unicode (ICU) libraries. In all other respects, the database is the same as the Windows version.

For more information about ICU and Windows Mobile, see "Installation considerations: Using ICU on Windows Mobile" on page 342.

The following sample applications are included with your SQL Anywhere for Windows Mobile installation:

- ADO.NET Sample
- ESQL Sample
- ODBC Sample

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can use these applications to access the sample database and examine the capabilities of SQL Anywhere for Windows Mobile.

Starting a SQL Anywhere database server on Windows Mobile

The SQL Anywhere Server icon is used to start the database server.

To start the SQL Anywhere server

1. From the **Start** menu, tap **Programs** » **SQLAny12**.
2. Tap **Server**.
3. In the **Database** field, type *\My Documents\demo.db* or tap **Browse** and locate *demo.db* in the *My Documents* directory.
4. The **Server Name** field may be left blank.

   The server name defaults to the database name *demo*.
5. The **Cache** field may be left unchanged.
6. The **Options** field may be left blank.
7. Tap **OK** to start the sample database running on the database server.
8. Navigate to the **Today** screen on your device.
9. Tap the database server icon located in the bottom right corner of the screen.

   When the message **Now accepting requests** appears in the database server messages window, you can proceed to the next lesson.

When you are finished using the sample database, you must shut down the database server.

To shut down the database server

1. Tap the network database server icon located in the bottom right corner of the Today screen.
2. On the menu, tap **Menu** » **Shut Down**.

The ADO.NET Sample

To use the ADO.NET Sample, you must have the Microsoft .NET Compact Framework version 2.0 or 3.0 installed on your device. The Microsoft .NET Compact Framework version 2.0 is included with Windows Mobile 6 devices, but not with Windows Mobile 5 devices. The ADO.NET Sample supports only Windows Mobile Classic and Professional devices with touch screens.

To download this component from the Microsoft Download Center, go to http://www.microsoft.com/downloads/search.aspx?displaylang=en.

The ADO.NET Sample demonstrates a simple application that uses the ADO.NET programming interface. This application allows you to start the sample database running on the network database server and access and modify data using SQL statements.

The source code for this sample is located in *samples-dir\SQLAnywhere\ce\ado_net_sample*.

You can load this project in Microsoft Visual Studio from *samples-dir\SQLAnywhere\ce\ado_net_sample\ado_net_sample.sln*.

Note
In the ADO.NET Sample user interface, SQL statements must be entered on a single line.

To use the ADO.NET Sample

1. Tap **Start** » **Programs** » **SQLAny12** » **ADO.NET Sample**.
2. Tap **Connect**.

   If the SQL Anywhere database server has not been started earlier, then tapping **Connect** will cause the server to start.
3. Tap **Exec SQL** to execute the default SQL statement, **SELECT * FROM Employees**.

   Data from the Employees table appears in the data window.
4. Navigate through the data in the Employees table using the scroll bars on the side and bottom of the data window.
5. Type the following query that accesses a more specific range of data:

   ```
   SELECT EmployeeID, Surname FROM Employees;
   ```
6. Tap **Exec SQL** to execute the SQL statement.

   The specified range of data replaces the data that was in the data pane.
7. Type **SELECT * FROM Employees ORDER BY EmployeeID** and tap **Exec SQL**.

   Notice the employee Matthew Cobb, with EmployeeID 105.
8. Type **UPDATE Employees SET Surname = 'Jones' WHERE Surname = 'Cobb'**, and then tap **Exec SQL** to execute the SQL statement.
9. Type **SELECT * FROM Employees ORDER BY EmployeeID** and tap **Exec SQL**.

   Notice that Matthew's last name has been changed from Cobb to Jones.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

10. Type **UPDATE Employees SET Surname = 'Cobb' WHERE Surname = 'Jones'** and then tap **Exec SQL** to reverse the change you made to the sample database.

11. Verify that the changes were reversed by typing **SELECT * FROM Employees ORDER BY EmployeeID** and then tapping **Exec SQL**.

    Notice that Matthew's last name has been changed back to Cobb.

12. Access data from another table by typing **SELECT * FROM Customers**, and then tapping **Exec SQL**.

    All the data from the Customers table appears in the data window, replacing the data from the Employees table.

13. Disconnect from the database server by tapping **Disconnect**.

    The ADO.NET Sample disconnects. The database server does not automatically shut down because the connection is pooled.

14. Close the ADO.NET Sample by tapping **x** in the top right corner of the window.

The ESQL Sample

The ESQL Sample demonstrates a simple application that uses the embedded SQL programming interface. This application allows you to start the sample database running on the network database server, and access data using SQL statements.

The source code for this sample can be found in *samples-dir\SQLAnywhere\ce\esql_sample*.

You can load this project file in Visual Studio 2005 from: *samples-dir\SQLAnywhere\ce\sql_sample \esql_sample.sln*.

Note
In the ESQL Sample user interface, SQL statements must be entered on a single line.

To use the ESQL Sample

1. Start the ESQL Sample by tapping **Start** » **Programs** » **SQLAny12** » **ESQL Sample**.

2. Tap **Connect** to connect to the sample database using the default connection string.

3. Tap **ExecSQL** to execute the default SQL statement, **SELECT * FROM Employees**.

   Data from the Employees table appears in the data window.

4. Use the scroll bars to view Employee table data.

5. To access data in the Customers table, type **SELECT * FROM Customers**, and tap **ExecSQL**.

Customer data replaces the employee data in the data window.

6. Tap **Menu** » **Disconnect** to shut down the network database server.

   The ESQL Sample disconnects and the network database server shuts down.

7. Close the ESQL Sample by tapping **x** in the top right corner of the window.

The ODBC Sample

The ODBC Sample demonstrates a simple application that uses the ODBC programming interface. This application allows you to start the sample database running on the network database server, and access data using basic SQL statements.

The source code for this sample can be found in *samples-dir\SQLAnywhere\ce\odbc_sample*.

You can load this project file in Visual Studio 2005 from: *samples-dir\SQLAnywhere\ce\odbc_sample\odbc_sample.sln*.

Note
In the ODBC Sample user interface, SQL statements must be entered on a single line.

To use the ODBC Sample

1. Start the ODBC Sample by tapping **Start** » **Programs** » **SQLAny12** » **ODBC Sample**.
2. Tap **Connect**.
3. Tap **ExecSQL** to execute the default SQL statement, **SELECT * FROM Employees**.

   Data from the Employees table appears in the data window.

4. Use the scroll bars to view Employee table data.
5. To access data in the Customers table, type **SELECT * FROM Customers** and tap **ExecSQL**.

   Customer data replaces the employee data in the data window.

6. Tap **Menu** » **Disconnect** to shut down the network database server.

   The ODBC Sample disconnects and the network database server shuts down.

7. Close the ESQL Sample.

Connecting to a database running on a Windows Mobile device

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

If you want to connect an application running on a desktop computer to a database running on a Windows Mobile device, you can connect over TCP/IP using the Windows Mobile Device Center (ActiveSync) link between the desktop computer and the Windows Mobile device. This allows you to administer a Windows Mobile database using the administration utilities on the desktop computer.

See also

- "Using the administration utilities on Windows Mobile" on page 363

Start a database server on your Windows Mobile device

If you want to connect from your desktop computer to a database server that is running on Windows Mobile, you must select the TCP/IP option when starting the server.

To start the database server on your Windows Mobile device for a remote connection

1. From the **Start** menu, tap **Programs** » **SQLAny12**.
2. Tap **Server**.
3. In the **Database** field, type the name of the database file that you want to start or click **Browse** to locate the database.

   By default, the sample database is located in *\My Documents\demo.db*.
4. In the **Server Name** field, type the database server name that you want to use.

   The default name for the sample database server is **demo**.
5. In the **Options** field, type **-x tcpip(port=2639)**.

   The port that the database server will use is 2639. It is best to avoid the use of port 2638 which is also the default TCP/IP port for a SQL Anywhere server running on your desktop system. For more information, see "-x dbeng12/dbsrv12 server option" on page 236.
6. Select **Use TCP/IP**.

   A TCP/IP connection is necessary to connect from a desktop computer to the database running on your Windows Mobile device.
7. Tap **OK** to start the sample database running on the network database server.

Now you can create an ODBC data source to connect from the desktop computer to your Windows Mobile device.

See also

- "Create an ODBC data source to connect to your Windows Mobile device" on page 350

Create an ODBC data source to connect to your Windows Mobile device

This section describes how to create an ODBC data source on your Windows desktop computer that connects to a database running on your Windows Mobile device.

For more information about ODBC data sources, see "Creating ODBC data sources" on page 98.

To create an ODBC data source to connect to your Windows Mobile device

1. On the Windows desktop computer, choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **ODBC Data Source Administrator**.
2. On the **User DSN** tab, click **Add**.
3. Select **SQL Anywhere 12**, and then click **Finish**.
4. On the **ODBC** tab, in the **Data Source Name** field, type **MobileServer**.
5. Click the **Login** tab.
   - **Authentication** Choose **Database**.
   - **User ID** Make sure this field is blank. Each time you connect to a database, you must supply a user ID and password.
   - **Password** Make sure this field is blank. The **Encrypt Password** box should be cleared.
   - **Action** Select **Connect To A Database On Another Computer**.
   - **Host** If you connected a Windows Mobile 5.0 or 6.0 device using Windows Mobile Device Center (ActiveSync) over a USB connection, try the Windows Mobile device IP address **169.254.2.1**.

     For some Windows Mobile devices, especially older devices, you might encounter difficulty using this address. If so, try the localhost (**127.0.0.1**) IP address. In this case you must also have set up a proxy port. For information on setting up a proxy port, see "Creating proxy ports for Windows Mobile devices" on page 351.

     For more information, see "Determine the IP address of your Windows Mobile device" on page 352.
   - **Port** Specify the port number that the Windows Mobile device listens on.

     Use the port that the database server on the Windows Mobile device is configured to use when it is started. It is recommended that you use port **2639**. See "Start a database server on your Windows Mobile device" on page 349.
   - **Server Name** Fill in the server name if you wish to connect to a specific database server running on the Windows Mobile device. Otherwise, make sure this field is blank.
   - **Database Name** Fill in the database name if you wish to connect to a specific database running on the Windows Mobile device. Otherwise, make sure this field is blank.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

6. Click **OK**.

You can now use this data source to connect from a Windows desktop computer to a database running on your Windows Mobile device.

To test the connection, you can use the dbping utility. If a database has been started on the Windows Mobile device, then use a command like the following to test the connection.

```
dbping -c "DSN=MobileServer;UID=DBA;PWD=sql"
```

For more information, see "Using the administration utilities on Windows Mobile" on page 363.

Creating proxy ports for Windows Mobile devices

To connect to a database server on a Windows Mobile device from the desktop, you might be required to configure a proxy port for Windows Mobile Device Center (ActiveSync) so that requests to the localhost (**127.0.0.1**) IP address are passed on to the database running on the Windows Mobile device.

To add a proxy port for a Windows Mobile device (Interactive SQL)

1. Start Interactive SQL on your desktop computer. For the 32-bit version of Windows Mobile Device Center (ActiveSync), you must start the 32-bit version of Interactive SQL.
2. At the bottom of the **Connect** window, click **Tools** and then choose **Setup Windows Mobile Proxy Port**.
3. Click **New**.
4. In the **Name** field, type **SQL Anywhere**.
5. In the **Port** field, type **2639**.

   This is the recommended TCP/IP port to use for a SQL Anywhere database server running on Windows Mobile. You may use other ports as long as you ensure that the database server on the Windows Mobile device is configured to use the same port.

   Note
   Every time your Windows Mobile device is cradled, Windows Mobile Device Center (ActiveSync) forwards traffic on port 2639 to the device. If you start a database server on your desktop computer while your Windows Mobile device is cradled, it cannot use port 2639. If this is problematic, you can choose another port to dedicate to Windows Mobile traffic.

6. Click **OK**.

   When setting up or changing a proxy port, it may be necessary to log off and then log in to your Windows desktop in order for the new proxy port setting to take effect.

You can also configure a proxy port by setting the following registry key: **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows CE Services\ProxyPorts**. Name the DWORD value **SQL**

Anywhere and specify the port dedicated to Windows Mobile traffic (2639 is a good choice). Note that for 64-bit versions of Windows running a 32-bit version of Windows Mobile Device Center (ActiveSync), this registry entry will be in the 32-bit hive.

Caution
Modifying the registry is dangerous. Modify the registry at your own risk.

Determine the IP address of your Windows Mobile device

When connecting to a database that is running on Windows Mobile, you may need the IP address to establish the connection.

To determine the IP address of your Windows Mobile device

1. From the **Start** menu, tap **Programs** » **SQLAny12**.
2. Tap **Server**.
3. In the **Database** field, type the name of the database file that you want to start or click **Browse** to locate the database.

   By default, the sample database is located in *\My Documents\demo.db*.
4. In the **Server Name** field, type the database server name that you want to use.

   The default name for the sample database server is **demo**.
5. In the **Options** field, type **-z -x tcpip(port=2639)**.

   With the -z option, the database server displays its IP address during startup. The address may change if you disconnect your Windows Mobile device from the network and then re-connect it. For more information, see "-z dbeng12/dbsrv12 server option" on page 243.

   The port that the database server will use is 2639. It is best to avoid the use of port 2638 which is also the default TCP/IP port for a SQL Anywhere server running on your desktop system. For more information, see "-x dbeng12/dbsrv12 server option" on page 236.
6. Select **Use TCP/IP**.

   A TCP/IP connection is necessary to connect from a desktop computer to the database running on your Windows Mobile device.
7. Tap **OK** to start the sample database running on the network database server.
8. Navigate to the **Today** screen on your device.
9. Tap the database server icon located in the bottom right corner of the screen.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The IP address appears in the database server messages window. Look at the first **Starting UDP listener on IP address** message in the list for the IP address.

For more information, see "Create an ODBC data source to connect to your Windows Mobile device" on page 350.

Configuring Windows Mobile databases

Most SQL features available in the full version of SQL Anywhere are also available in the Windows Mobile version. These include transaction processing, referential integrity actions, procedures and triggers, and so on. However, the Java features and the remote data access features are not available on Windows Mobile.

You should be mindful of the unsupported features when creating a database intended for a Windows Mobile device.

For a complete list of unsupported features, see "SQL Anywhere feature support on Windows Mobile" on page 370.

The following settings are configured during database creation. Once set, these properties can only be changed by rebuilding the database.

- **Case sensitivity or insensitivity** See "Case sensitivity" [*SQL Anywhere Server - SQL Usage*].
- **Treatment of trailing blanks in comparisons** By default, databases are created with trailing blanks classified as extra characters. For example, 'Dirk' is not the same as 'Dirk '. You can create databases with blank padding, so that trailing blanks are ignored. See "Ignore trailing blanks in comparisons" [*SQL Anywhere Server - SQL Usage*].
- **Page size** See "Table and page sizes" [*SQL Anywhere Server - SQL Usage*].
- **Collation sequence and character set** When creating databases for Windows Mobile, you should use a collation based on the same single- or multibyte character set that Windows would use for the language of interest. For example, if you are using English, French, or German, use the 1252Latin1 collation. If you are using Japanese, use the 932JPN collation, and if you are using Korean, use the 949KOR collation. See "Understanding collations" on page 419.

Note
Do not specify a locale or a sorttype in the tailoring options string when creating a database for use on Windows Mobile. If you do, it is likely that the will not start on the Windows Mobile device. For more information about collation tailoring options, see "Collation tailoring options" on page 427.

Because character set translation is not supported on Windows Mobile, you must use either the operating system character set or UTF-8 for Windows Mobile databases.

You must choose whether you want to install the ICU library when creating your Windows Mobile database. See "Installation considerations: Using ICU on Windows Mobile" on page 342.

Using a transaction log on Windows Mobile

The transaction log stores all changes made to a database, in the order in which they are made. In the event of media failure on a database file, the transaction log is essential for database recovery. It also makes your work more efficient. By default, the transaction log is placed in the same directory as the database file. It is created when the database is started for the first time on your Windows Mobile device.

When you copy an existing database to your Windows Mobile device, you can copy both the database and transaction log files. If you do not copy the transaction log file to the device, a new transaction log is created when you start the database on your Windows Mobile device. The new transaction log does not contain the information contained in the original transaction log. This can be problematic if the database was not shut down properly the last time it was used, or if the database is involved in synchronization. The best practice is to copy both the database and the transaction log file to the Windows Mobile device.

See also

- "The transaction log" on page 21

Using jConnect on Windows Mobile

Sybase jConnect is a pure Java JDBC driver that can be used with SQL Anywhere. Sybase Central and Interactive SQL give you the option to include jConnect metadata support so that you can use the jConnect JDBC driver to access SQL Anywhere databases.

By default, jConnect metadata support is not enabled by the **Create Database Wizard** for databases being created for Windows Mobile. However, you can choose to enable jConnect metadata support if you require it.

Adding jConnect metadata support to a database adds many entries to the system tables. This adds to the size of the database and, more significantly, adds about 200 KB to the memory requirements for running the database, even if you do not use any jConnect functionality.

If you are not going to use jConnect, and you are running in a limited-memory environment like Windows Mobile, it is recommended that you do not add jConnect metadata support to your database.

See also

- "Using the jConnect JDBC driver" [*SQL Anywhere Server - Programming*]

Using encryption on Windows Mobile

You can choose to secure your database either with simple or strong encryption. The only way to change the encryption setting after a database has been initialized is by rebuilding the entire database.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Encrypting and decrypting a database" on page 1130
- "Keeping your Windows Mobile database secure" on page 1141

Creating a Windows Mobile database

You can create a SQL Anywhere database for your Windows Mobile device:

- With the Sybase Central **Create Database Wizard** to create a database that can be copied directly to your Windows Mobile device.
- With the Initialization utility (dbinit) to create a database that can be copied manually to your Windows Mobile device.
- With the CREATE DATABASE statement in Interactive SQL to create a database that can be copied manually to your Windows Mobile device.

Note
If you want the **Create Database Wizard** to automatically copy a new database to your Windows Mobile device, then Sybase Central must be the same bitness as ActiveSync or Windows Mobile Device Center (32-bit or 64-bit). For Windows XP and earlier, only 32-bit software is supported.

For information about decisions you need to make creating a Windows Mobile database, see:

- "Using a transaction log on Windows Mobile" on page 354
- "Installation considerations: Using ICU on Windows Mobile" on page 342
- "Installation considerations: Using the .NET Compact Framework on Windows Mobile" on page 343

Note
When you run a database on Windows Mobile, the database server automatically turns on write checksums. This behavior helps to provide early detection if the database file becomes corrupt. See "Using checksums to detect corruption" on page 928.

Create a Windows Mobile database using Sybase Central

Sybase Central has features to make database creation easy for Windows Mobile. Sybase Central enforces the requirements for Windows Mobile databases, and gives you the option of copying the database file to your device.

To create a Windows Mobile database in Sybase Central and copy it directly to your Windows Mobile device

1. Connect your Windows Mobile device to your desktop computer.

2. Start Sybase Central on your desktop computer. For the 32-bit version of Windows Mobile Device Center (ActiveSync), you must start the 32-bit version of Sybase Central. Sybase Central will use Windows Mobile Device Center (ActiveSync) to transfer the database to the Windows Mobile device.

3. Choose **Tools** » **SQL Anywhere 12** » **Create Database**.

4. Click **Create A Database On This Computer**. Click **Next**.

5. Specify a file name and directory to store the database file in on your desktop computer, and then click **Next**.

6. Select **Create This Database For Windows Mobile** and then click **Next**.

7. Select **Copy The Database To Your Windows Mobile Device** and then click **Next**.

8. Specify the Windows Mobile directory to copy your database files to. The default location is the main device directory.

   Tip
   Copy the database to the *My Documents* directory of your Windows Mobile device to make it simpler to start the database.

   When starting a database on your Windows Mobile device using the **Server Startup Options** window, you can only use **Browse** to search for the database file in the *My Documents* directory.

   If the database is not stored in the *My Documents* directory, you must type the path of the database in the **Database** field of the **Server Startup Options** window.

   Optionally, you can select the **Delete The Desktop Database After Copying** option.

   If you choose not to delete the desktop computer copy, a copy of the database file is stored on your desktop computer in the directory that you specified in Step 5. Click **Next**.

9. Specify the directory where you want to save the transaction log file. Click **Next**.

   On your Windows Mobile device, the transaction log file is created in the same directory as the database file when the database is started on the network database server for the first time.

10. Specify whether you want to use a transaction log mirror. Click **Next**.

11. Clear the **Install jConnect Metadata Support** option and then click **Next**.

12. Set the level of encryption for your database by selecting the appropriate option and then click **Next**.

    If you select strong encryption, you must specify an encryption key. It is recommended that you choose a value for your key that is at least 16 characters long, contains a mix of upper and lowercase, and includes numbers, letters, and special characters.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Caution
Be sure to store a copy of your key in a safe location. You require the key each time you want to start or modify the database. A lost key will result in a completely inaccessible database, from which there is no recovery.

13. Select a page size and click **Next**.

14. On the **Specify Additional Settings** page, select **Include Checksum With Each Database Page** and then click **Next**.

15. Follow the remaining instructions in the wizard and then click **Finish** to create the database and copy it to your device.

Specify a collation sequence for NCHAR data
If NCHAR UCA sorting is not required, the NCHAR collation sequence should be UTF8BIN. In this way, the ICU libraries (*dbicu12.dll* and *dbicudt12.dll*) are not required by the database server. Select **Use The Following Supplied Collation**, and then select **UTF8BIN**.

16. A window appears, tracking the progress of the files being copied to your Windows Mobile device. Click **Close**.

17. Once the wizard has copied the database to your Windows Mobile device, verify the location of the files:

    From the **Start** menu, tap **Programs** » **File Explorer** and navigate to the Windows Mobile directory that you copied the database to.

    The database file is listed there. The transaction log file does not appear until the first time you start the database on your Windows Mobile device.

Create a Windows Mobile database using dbinit

The Initialization utility (dbinit) can be used to create databases that can be used on Windows Mobile. However, you cannot copy them directly to a Windows Mobile device from this utility. You must manually copy databases created with the dbinit utility to your Windows Mobile device.

To create a database using the dbinit utility

1. At a command prompt, navigate to the directory where you want to create your database. For example:

   ```
   cd temp
   ```

2. Create your database by running the following command:

   ```
   dbinit -s database-name.db
   ```

   The -s option enables checksums for the database.

Tip
You can also configure database properties such as encryption and page size using the Initialization utility. See "Initialization utility (dbinit)" on page 799.

3. Copy the database to your Windows Mobile device.

   For more about copying the database to your Windows Mobile device, see "Copy a database to your Windows Mobile device" on page 359.

Create a Windows Mobile database using the CREATE DATABASE statement

The CREATE DATABASE statement can be used to create databases in Interactive SQL on your desktop computer. However, you cannot copy them directly to a Windows Mobile device from this application. You must manually copy databases to your Windows Mobile device.

To create a database using the CREATE DATABASE statement

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Interactive SQL**.

   If the **Connect** window does not appear automatically, choose **SQL** » **Connect**.

2. Click the **Identification** tab and complete the following fields:
   - **User ID** Type **DBA**.
   - **Password** Type **sql**.
   - **Action** Select **Connect With An ODBC Datasource**.
   - **ODBC Data Source Name** Click **Browse**, choose the **MobileServer** data source that you created in "Create an ODBC data source to connect to your Windows Mobile device" on page 350.

3. Type the following statement in the **SQL Statements** pane of Interactive SQL:

```
CREATE DATABASE 'c:\\temp\\database-name.db'
TRANSACTION LOG ON
CHECKSUM ON;
```

Tip
You can also configure database properties such as encryption and page size using the CREATE DATABASE statement. See "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

4. From the **SQL** menu, choose **Execute**.

   A database and transaction log are created in the *c:\temp* directory of your desktop computer.

5. Copy the database to your Windows Mobile device.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For information about copying the database to your Windows Mobile device, see "Copy a database to your Windows Mobile device" on page 359.

Copy a database to your Windows Mobile device

Any existing SQL Anywhere database can be copied to your Windows Mobile device using the method described in this section. However, you must keep in mind that any database features that are not supported on Windows Mobile do not work when you copy the database to your Windows Mobile device. See "SQL Anywhere feature support on Windows Mobile" on page 370.

To copy a database to your Windows Mobile device

1. Connect the Windows Mobile device to your desktop computer.
2. Open Windows Explorer on your desktop computer.
3. Browse to the directory on your desktop computer containing the database that you want to copy.
4. Right-click the database file and choose **Copy**.
5. Browse to the directory on your Windows Mobile device where you want to store the database file.

   Tip
   When starting a database on your Windows Mobile device using the **Server Startup Options** window, you can only use **Browse** to search for the database file in the *My Documents* directory and its subdirectories.

   If the database is not stored in the *My Documents* directory, you must type the path of the database in the **Database** field of the **Server Startup Options** window.

6. Right-click an open area of the Windows Explorer window for your Windows Mobile device and then choose **Paste**.

   The file is copied to the Windows Mobile device.

Rebuilding databases on Windows Mobile

When rebuilding a database on Windows Mobile, you have the following options:

- Rebuild the Windows Mobile database on another platform and then copy the database to the Windows Mobile device. This is the recommended method.
- Repopulate an empty database using dbmlsync.
- Repopulate an empty database using dbremote.

- Use dbunload on the Windows Mobile device. This option is not available on smartphones.

The first three options are recommended when upgrading a Windows Mobile database. However, if these options are not available to you, you can use dbunload on Windows Mobile. Before deciding to use dbunload on Windows Mobile, you should consider the following implications of using dbunload on Windows Mobile:

- the size of the database server's temporary file (both the unload and reload can cause this file to grow to several megabytes)
- the extra space required for dbunload and related components
- the extra cost of having multiple copies of a database on the Windows Mobile device

Because running dbunload on a Windows Mobile device can require more resources than some devices have available, upgrading the database on a different platform is recommended whenever possible.

Note
If you want to run dbunload on a Windows Mobile device, you must choose the **Unload/Reload Support** option in the **Deploy SQL Anywhere 12 for Windows Mobile Wizard**. You can modify your SQL Anywhere installation to add this support if you did not select this option when you first installed SQL Anywhere for Windows Mobile.

Notes about using dbunload on Windows Mobile

To use dbunload on a Windows Mobile device, ensure you have performed the following tasks:

- The following files should be deployed to your SQL Anywhere installation directory (by default, *\Program Files\SQLAny12*):
  - *dbsrv12.exe*
  - *dbunlspt.exe*
  - *dbunload.exe*
  - *dbrunsql.exe*
- The following files should be deployed to the *\Windows* directory:
  - *dblgen12.dll*
  - *dblib12.dll*
  - *dbscript12.dll*
  - *dbtool12.dll*
  - *dbusen.dll*
- The following registry entry string value should be set to the SQL Anywhere software directory: *HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\SQL Anywhere\12.0\Location*.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Rebuild a database on Windows Mobile

The following steps can be embedded into third-party Windows Mobile applications so that the process is automated for the end user. If you choose to do this, then you should consider using the -qc and/or -q dbunload and dbrunsql options or calling the DBUnload function in *dbtool12.dll*.

To unload a database on Windows Mobile (dbunload)

1. On a platform other than Windows Mobile, create a new, empty SQL Anywhere 12 database.

   The CHAR collation sequence should match that of the existing database. If NCHAR UCA sorting is not required, the NCHAR collation sequence should be UTF8BIN. In this way, the ICU libraries (*dbicu12.dll*, *dbicudt12.dll*) are not required by the database server.

2. Copy the SQL Anywhere 12 software and the empty SQL Anywhere database file to the Windows Mobile device. See "Notes about using dbunload on Windows Mobile" on page 360.

3. Ensure there are no database servers running on the device.

4. Run the following command:

   ```
   dbunload-path\dbunload -c "UID=DBA;PWD=DBA-
   password;CHARSET=none;DBF=existing-database" unload-directory
   ```

5. Ensure that dbunload succeeded, and then close the dbunload window.

6. Run the following command:

   ```
   dbrunsql-path\dbrunsql -c "UID=DBA;PWD=sql;CHARSET=none;DBF=new-empty-
   SQLAnywhere12databasefile" -g- \reload.sql
   ```

7. Ensure that dbrunsql succeeded, and then close the dbrunsql window.

8. Remove the *reload.sql* file and *unload-directory* from the Windows Mobile device.

Backing up a Windows Mobile database

Backup and recovery is vital to ensure you do not lose data in the event of data corruption or media failure. It is best to back up your Windows Mobile database to a physically separate location to safeguard against data loss because of theft or loss of the device, or media failure on the device.

Most backup and recovery utilities are available on Windows Mobile. However, these utilities are not useful since you cannot use the utilities on Windows Mobile to store backups in a physically separate location (with the exception of backing up main memory databases to removable SD cards). Instead, data can be backed up by copying the entire database file to a desktop computer. You can also use synchronization to maintain an up-to-date copy of your Windows Mobile database on a desktop computer. See "Understanding MobiLink synchronization" [*MobiLink - Getting Started*].

Erase a Windows Mobile database

SQL Anywhere for Windows Mobile does not support the **Erase Database Wizard**, the DROP DATABASE statement, or the Erase utility (dberase). You must manually erase databases from your Windows Mobile device. The database must not be running when you erase it.

There are two methods for erasing a database from your Windows Mobile device. You can erase a database through the device interface, or you can connect your device to a desktop computer and erase the database using Windows Explorer.

After you delete the database, delete the transaction log file, if one exists.

To erase a database using the device interface

1. From the **Start** menu, tap **Programs** » **File Explorer** and navigate to the directory containing the database file that you want to erase.
2. Tap and hold the database file.
3. Tap **Delete**.
4. Tap **Yes** to confirm the deletion.

To erase a database using Windows Explorer

1. Place your Windows Mobile device in its cradle and ensure that it connects to the desktop computer via Windows Mobile Device Center (ActiveSync).
2. Open Windows Explorer on your desktop computer.
3. Browse to the Windows Mobile directory where the database file is stored.
4. Right-click the database file and choose **Delete**.
5. Click **Yes**.

Running the database server on Windows Mobile

The Windows Mobile database server does not start the TCP/IP network link unless it is explicitly requested.

On Windows Mobile, attempting to start a second SQL Anywhere database server while a first database server is already running brings the first server to the foreground. This is standard behavior for Windows Mobile applications. Because of this behavior, you cannot run two database servers at the same time on a Windows Mobile device. However, SQL Anywhere supports running multiple databases on a single database server.

For more information about starting a database server on Windows Mobile, see "Tutorial: Running Windows Mobile databases from Sybase Central" on page 363.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Specifying server options on Windows Mobile

You can specify server and database options when starting the database server to tune SQL Anywhere behavior and performance. You can choose from many options to specify such features as how much memory the cache can use, the level of permission needed to start a database on the database server, and the network protocols to use.

On Windows Mobile, options are specified in the **Server Startup Options** window. This is different than other Windows operating systems where database server options can be set on the command line. Most server options are available for Windows Mobile.

For more information about database server options, see "The SQL Anywhere database server" on page 147.

For information about unsupported options, see "Database server option support on Windows Mobile" on page 373.

Using the administration utilities on Windows Mobile

This section describes specific considerations for using the SQL Anywhere database administration utilities with Windows Mobile databases.

Tutorial: Running Windows Mobile databases from Sybase Central

Sybase Central is a database management tool that provides a graphical user interface for administering SQL Anywhere. Sybase Central can also be used for managing other products, including MobiLink synchronization.

Once you complete this tutorial, you will be able to perform key tasks associated with the database server: starting and stopping the server, running single and multiple databases on a database server, and connecting to a database.

Requirements

- Complete all the tasks in the following sections before you begin the tutorial:
  - "Connecting to a database running on a Windows Mobile device" on page 348
  - "Create an ODBC data source to connect to your Windows Mobile device" on page 350
- Connect your Windows Mobile device to a desktop computer.

Before you begin

You need to create two Windows Mobile databases for use in the tutorial.

To create databases for your Windows Mobile device (Sybase Central)

1. Connect your Windows Mobile device to the desktop computer.

2. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
3. Choose **Tools** » **SQL Anywhere 12** » **Create Database**.
4. Follow the instructions in the **Create Database Wizard**.
5. On the **Specify A Database File** page, click **Browse** and select a location for the database file. Name the database file *Alpha*. Click **Save**.
6. On the **Choose To Create For Windows Mobile** page, click **Create This Database For Windows Mobile** and then click **Next**.

   The wizard tests the connection to your Windows Mobile device.
7. On the **Choose To Copy The Database** page, select **Copy The Database To Your Windows Mobile Device**.
8. In the **Windows Mobile File Name** field, type *\My Documents\Alpha.db*.
9. Select **Delete The Desktop Database After Copying** and then click **Next**.
10. On the **Specify A Collation Sequence For NCHAR Data** page, select **Use The Following Supplied Collation**, and then select **UTF8BIN**.

    For more information, see "Installation considerations: Using ICU on Windows Mobile" on page 342.
11. Click **Finish**.
12. Click **Close**.
13. Repeat this procedure and create a database called *\My Documents\Beta.db*.

Lesson 1: Start the database server

This section describes the simple case of running a single database on Windows Mobile.

To start a database on the server

1. From the **Start** menu, tap **Programs** » **SQLAny12**.
2. Tap **Server**.
3. In the **Database** field, type the name of the database file that you want to start or tap **Browse** and locate the *Alpha.db* file in the *My Documents* directory.
4. In the **Server Name** field, type **MobileServer**.
5. In the **Options** field, type **-gd all -x tcpip(port=2639)**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The -gd option sets the permissions to allow any user to start additional databases on the network database server. This is necessary in a later lesson. See "-gd dbeng12/dbsrv12 server option" on page 185.

The port that the database server will use is 2639. It is best to avoid the use of port 2638 because it is the default TCP/IP port for a SQL Anywhere server running on your desktop system. For more information, see "-x dbeng12/dbsrv12 server option" on page 236.

6. Select **Use TCP/IP**.

   A TCP/IP connection is necessary to connect from a desktop computer to the database running on your Windows Mobile device.

7. Tap **OK** to start the sample database running on the network database server.

8. Navigate to the **Today** screen on your device.

9. Tap the database server icon located in the bottom right corner of the screen.

   When the message **Now accepting requests** appears in the database server messages window, you can proceed to the next lesson.

What's next?

Next, you will learn how to start multiple databases on the network database server on Windows Mobile.

Lesson 2: Start multiple databases on the Windows Mobile database server

On Windows Mobile devices, attempting to start a second SQL Anywhere database server while a first database server is already running brings the first database server to the foreground. This is standard behavior for Windows Mobile applications. Because of this behavior, two database servers cannot run at the same time on a Windows Mobile device. As an alternative to running multiple database servers, one server can run multiple databases.

To connect to a database from Sybase Central

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.

2. Choose **Connections** » **Connect With SQL Anywhere 12**.

3. In the **Authentication** dropdown list, choose **Database**.
   a. In the **User ID** field, type **DBA**.
   b. In the **Password** field, type **sql**.

4. From the **Action** dropdown list, choose **Connect To A Running Database On Another Computer**.
   a. In the **Host** field, type **169.254.2.1**.
   b. In the **Port** field, type **2639**.

c. In the **Server Name** field, type **MobileServer**.

d. In the **Database Name** field, type **Alpha**.

5. Click **Browse** and choose the **MobileServer** data source that you created in "Create an ODBC data source to connect to your Windows Mobile device" on page 350.

6. Click **Connect** to connect to the *Alpha.db* database running on your Windows Mobile device.

7. If you fail to connect to the database server, see "Create an ODBC data source to connect to your Windows Mobile device" on page 350.

Now that you have started the database server and connected to the Alpha database, you can start additional databases on your Windows Mobile device.

To start a second database on the network database server

1. In the left pane of Sybase Central, right-click **MobileServer** and choose **Start Database**.

2. In the **Database File** field, type **\My Documents\Beta.db**.

3. Click **OK** to start the database on the network database server.

The database is loaded on the network database server. Now you must initiate a connection from your desktop computer.

To connect to the second database

1. In the left pane of Sybase Central, right-click **Beta** and choose **Connect**.

2. In the **Authentication** dropdown list, choose **Database** then:

   a. In the **User ID** field, type **DBA**.

   b. In the **Password** field, type **sql**.

3. From the **Action** dropdown list, choose **Connect To A Running Database On Another Computer**.

   a. In the **Host** field, type **169.254.2.1**.

   b. In the **Port** field, type **2639**.

   c. In the **Server Name** field, type **MobileServer**.

   d. In the **Database Name** field, type **Beta**.

4. Click **Connect** to connect to the Beta database running on your Windows Mobile device.

You can now view and manipulate the data in the Alpha and Beta databases using Sybase Central.

What's next?

Next, you will learn how to disconnect from the databases and shut down the database server on Windows Mobile.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Lesson 3: Shut down the database server on Windows Mobile

Before you can shut down the network database server on your Windows Mobile device, you must stop the connections from your desktop computer.

To disconnect from the Windows Mobile databases

1. In Sybase Central, choose **Connections** » **Disconnect**.
2. Select the connection that corresponds to the Alpha database.
3. Click **OK**.
4. Choose **Connections** » **Disconnect**.

   The Alpha database is disconnected.
5. Repeat steps 1-4 for the Beta database.

Now that you have disconnected from the Windows Mobile databases in Sybase Central, you can shut down the network database server.

To shut down the server

1. On the Windows Mobile device, tap the database server icon located in the bottom right corner of the **Today** screen.
2. Tap **Menu** » **Shut Down**.

Where do I go from here?

Once you connect to a database from Sybase Central, you can add data to the tables in your database, add and edit database objects, and perform other administrative tasks.

For information about administering databases from Sybase Central, see:

- "Using the SQL Anywhere 12 plug-in" on page 691
- "Working with database objects" [*SQL Anywhere Server - SQL Usage*]

Tutorial: Managing Windows Mobile databases with Interactive SQL

Interactive SQL is an application that allows you to query and alter data in your database, and modify the structure of your database. Interactive SQL provides a pane for you to enter SQL statements, and panes that display information about how the query was processed and the result set.

This tutorial provides a brief introduction to using Interactive SQL from a desktop computer to manage databases on your Windows Mobile device. You will learn how to connect to the sample database on your Windows Mobile device from Interactive SQL. Once connected, you can use Interactive SQL to execute SQL statements.

Lesson 1: Start the sample database

The sample database must be running on your Windows Mobile device before you can connect to it from Interactive SQL.

To start the sample database

1. From the **Start** menu, tap **Programs** » **SQLAny12**.
2. Tap **Server**.
3. In the **Database** field, type the path of the sample database. The default location is *\My Documents \demo.db*. Alternately, use **Browse** to locate the *demo.db* file.
4. In the **Server Name** field, type **MobileServer**.
5. In the **Cache** field, type **5MB**.

   The default cache size on Windows Mobile is 600 KB. However, a larger cache size is recommended because it can help improve performance.
6. In the **Options** field, type **-x tcpip(port=2639)**.

   The port that the database server will use is 2639. It is best to avoid the use of port 2638 which is also the default TCP/IP port for a SQL Anywhere server running on your desktop system. For more information, see "-x dbeng12/dbsrv12 server option" on page 236.
7. Select **Use TCP/IP**.

   A TCP/IP connection is necessary to connect from a desktop computer to the database running on your Windows Mobile device.
8. Tap **OK** to start the sample database running on the network database server.
9. Navigate to the **Today** screen on your device.
10. Tap the database server icon located in the bottom right corner of the screen.

    When the message **Now accepting requests** appears in the database server messages window, you can proceed to the next lesson.

What's next?

Next you will learn how to connect from Interactive SQL to the database running on your Windows Mobile device.

Lesson 2: Start Interactive SQL and connect

Now that the sample database is running on your Windows Mobile device, connect to it from Interactive SQL to view and manage the database from your desktop computer.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To connect from Interactive SQL to a database on your Windows Mobile device

1. On the desktop computer, choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Interactive SQL**.
2. In the **Authentication** dropdown list, choose **Database** then:
   a. In the **User ID** field, type **DBA**.
   b. In the **Password** field, type **sql**.
3. From the **Action** dropdown list, choose **Connect To A Running Database On Another Computer**.
   a. In the **Host** field, type **169.254.2.1**.
   b. In the **Port** field, type **2639**.
   c. In the **Server Name** field, type **MobileServer**.
   d. In the **Database Name** field, type **Beta**.
4. Click **Connect** to connect to the sample database running on your Windows Mobile device.

   If you fail to connect to the database server, see "Create an ODBC data source to connect to your Windows Mobile device" on page 350.

What's next?

You can now view and manage the data in the sample database from Interactive SQL.

Lesson 3: Execute queries against a Windows Mobile database

One of the principal uses of Interactive SQL is to browse table data. Interactive SQL retrieves information by sending a request to your database server. The database server, in turn, looks up the information, and returns it to Interactive SQL.

To execute a SQL statement against a Windows Mobile database

1. In the SQL Statements pane, execute the following statement:

   ```
   SELECT * FROM Employees;
   ```

2. From the **SQL** menu, choose **Execute** to execute the statement.

   All the data from the Employees table appears in the **Results** pane.

3. From the **SQL** menu, choose **Disconnect** to disconnect from the Windows Mobile database.

Where do I go from here?

Once you connect to a database from Interactive SQL, you can view and manipulate the data, and add and modify database objects.

For information about Interactive SQL, writing queries, and using SQL statements, see:

- "Using Interactive SQL" on page 697
- "Querying data" [*SQL Anywhere Server - SQL Usage*]
- "Summarizing, grouping, and sorting query results" [*SQL Anywhere Server - SQL Usage*]
- "Joins: Retrieving data from several tables" [*SQL Anywhere Server - SQL Usage*]
- "SQL statements" [*SQL Anywhere Server - SQL Reference*]

SQL Anywhere feature support on Windows Mobile

This section lists the components and features of SQL Anywhere that are unsupported or have altered functionality on Windows Mobile. Where available, alternatives to unsupported features are listed.

For more information about supported and unsupported components on Windows Mobile, see http://www.sybase.com/detail?id=1061806.

SQL Anywhere includes several tools for administering databases. These include Sybase Central, Interactive SQL, and utilities. None of these administration tools can be deployed to Windows Mobile. Instead, database administration is performed from a Windows-based desktop computer that is connected to the Windows Mobile device.

For more information, see "Using the administration utilities on Windows Mobile" on page 363.

Component or feature	Considerations
Application profiling	When you create a tracing session for a database running on Windows Mobile, you must configure tracing using the **Database Tracing Wizard** (you cannot use the **Application Profiling Wizard**). As well, you must trace data from the Windows Mobile device to a copy of the Windows Mobile database running on a database server on a desktop computer. You cannot automatically create a tracing database from a Windows Mobile device, and you cannot trace to the local database on a Windows Mobile device. See "Application profiling" [*SQL Anywhere Server - SQL Usage*].
Database mirroring	Unsupported on Windows Mobile.
External stored procedures	Unsupported on Windows Mobile.
SQL Anywhere JDBC driver	Unsupported on Windows Mobile. You can use jConnect on Windows Mobile.
External environments	Unsupported on Windows Mobile.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Component or feature	Considerations
jConnect	The jConnect driver can be enabled when you create a database for Windows Mobile. This can be useful if you want to move the database to a desktop computer that supports Java. However, enabling the jConnect driver adds to the size of the database and, more significantly, adds about 200 KB to the memory requirements for running the database. This additional memory requirement should be considered when running the database in a limited-memory environment like Windows Mobile. See "Using jConnect on Windows Mobile" on page 354.
Kerberos authentication	Unsupported on Windows Mobile.
LDAP	Unsupported on Windows Mobile.
ODBC clients	Windows Mobile does not provide an ODBC driver manager or an ODBC administrator, so SQL Anywhere uses ODBC data sources stored in files. See "Using ODBC data sources on Windows Mobile" on page 104.
Sybase Open Client	Unsupported on Windows Mobile.
Parallel backups	Unsupported on Windows Mobile.
Personal database server (dbeng12)	Only the network database server (*dbsrv12.exe*) is supported on Windows Mobile. This executable supports local connections and client/server communications across a network.
Remote data access (including directory access servers)	Unsupported on Windows Mobile.

SQL statement support on Windows Mobile

This section describes SQL statements that are not supported on Windows Mobile, and those that have altered or limited functionality.

For a complete list of SQL statements, see "SQL statements" [*SQL Anywhere Server - SQL Reference*].

SQL statement	Considerations
BACKUP statement	Only the BACKUP DATABASE DIRECTORY clause is supported on Windows Mobile.

SQL statement	Considerations
CREATE DATABASE statement	The CREATE DATABASE statement can be used to initialize a database on a desktop computer, which can later be copied to a Windows Mobile device. See "Creating a Windows Mobile database" on page 355.
CREATE EVENT statement	DiskSpace event types are not supported on Windows Mobile. However, you can use this statement to define the GlobalAutoincrement event type or the ServerIdle event type.
CREATE EXISTING TABLE statement	Unsupported on Windows Mobile.
CREATE EXTERNLOGIN statement	Unsupported on Windows Mobile.
CREATE FUNCTION statement	The CREATE FUNCTION statement can be used on Windows Mobile to create user-defined SQL functions for use in the database. Note that the EXTERNAL NAME clause is not supported on Windows Mobile.
CREATE SERVER statement	Unsupported on Windows Mobile.
CREATE TABLE statement	The AT clause of the CREATE TABLE statement for creating proxy tables is not supported on Windows Mobile.
DROP DATABASE statement	Unsupported on Windows Mobile.
DROP SERVER statement	Unsupported on Windows Mobile.
INSTALL JAVA statement	Unsupported on Windows Mobile.
REMOVE JAVA statement	Unsupported on Windows Mobile.
REORGANIZE TABLE statement	Unsupported on Windows Mobile.
RESTORE DATABASE statement	Unsupported on Windows Mobile.
START JAVA statement	Unsupported on Windows Mobile
STOP JAVA statement	Unsupported on Windows Mobile.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Database server option support on Windows Mobile

This section describes those database server options that are not supported or have altered functionality on Windows Mobile.

Option	Considerations
@data option	Unsupported on Windows Mobile.
-? server option	Unsupported on Windows Mobile.
-cm server option	Unsupported on Windows Mobile.
-cw server option	Unsupported on Windows Mobile.
-ec server option	Strong communication encryption (TLS) is not supported on Windows Mobile. Only the **none** and **simple** settings are supported. See "-ec dbeng12/dbsrv12 server option" on page 176.
-gb server option	Unsupported on Windows Mobile.
-ge server option	Unsupported on Windows Mobile.
-gn server option	Unsupported on Windows Mobile.
-gna server option	Unsupported on Windows Mobile.
-gnh server option	Unsupported on Windows Mobile.
-gnl server option	Unsupported on Windows Mobile.
-gns server option	Unsupported on Windows Mobile.
-gt server option	Unsupported on Windows Mobile.
-gtc server option	Unsupported on Windows Mobile.
-qi server option	When running, the network database server appears as an icon in the bottom right corner of the Today screen on your Windows Mobile device. This feature cannot be disabled.
-s server option	Unsupported on Windows Mobile.
-tmf server option	Unsupported on Windows Mobile.
-tmt server option	Unsupported on Windows Mobile.
-u server option	Unsupported on Windows Mobile.

Option	Considerations
-ua server option	Unsupported on Windows Mobile.
-uc server option	Unsupported on Windows Mobile.
-ud server option	Unsupported on Windows Mobile.
-uf server option	Unsupported on Windows Mobile.
-ui server option	Unsupported on Windows Mobile.
-ut server option	Unsupported on Windows Mobile.
-ux server option	Unsupported on Windows Mobile.
-xp server option	Unsupported on Windows Mobile.
-ze server option	Unsupported on Windows Mobile.

Sybase Central wizard support on Windows Mobile

The following table lists the Sybase Central wizards that are not supported or have altered functionality on Windows Mobile and provides alternatives where possible.

Wizard	Considerations
Backup Database Wizard	Archive backups are not supported on Windows Mobile. The **Backup Database Wizard** is not supported. See "Types of backup" on page 888. As an alternative on Windows Mobile, you can use the **Create Backup Images Wizard**, which makes a separate backup of the database and transaction log files. See "Use the Backup Database Wizard" on page 897.
Change Log File Settings Wizard	Unsupported on Windows Mobile.
Create Database Wizard	This wizard provides options for creating a database for use on Windows Mobile, provided Windows Mobile services are installed on the desktop computer running Sybase Central. See "Creating a Windows Mobile database" on page 355.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Wizard	Considerations
Create Maintenance Plan Wizard	The following options are not available on Windows Mobile: • Full Archive Backup • Back up to Tape • Email the Maintenance Plan Report
Erase Database Wizard	Unsupported on Windows Mobile.
Migrate Database Wizard	Unsupported on Windows Mobile.
Restore Database Wizard	Unsupported on Windows Mobile.
Create Service Wizard	Unsupported on Windows Mobile.
Translate Log File Wizard	Unsupported on Windows Mobile.
Unload Database Wizard	This wizard cannot map to the Windows Mobile directory where the database files are stored. However, you can unload a Windows Mobile database by copying it to your desktop computer and using the **Unload Database Wizard**.
Upgrade Database Wizard	This wizard is not supported on Windows Mobile. However, you can upgrade a Windows Mobile database by copying it to your desktop computer and using this wizard before copying the database back to your Windows Mobile device. See "Upgrading SQL Anywhere" [*SQL Anywhere 12 - Changes and Upgrading*].

SQL Remote support on Windows Mobile

SQL Remote is supported on Windows Mobile with the following exceptions:

Component or feature	Considerations
Extraction utility (dbxtract)	Windows Mobile does not support this utility. If necessary, a Windows Mobile database can be copied to a desktop computer so that the Extraction utility can be used.

Configuring your database

This section describes the files used by SQL Anywhere, database limitations, and how to configure database properties and options. It also describes how to configure your SQL Anywhere installation to handle international language issues.

SQL Anywhere environment variables

SQL Anywhere uses environment variables to store various types of information. Not all environment variables need to be set in all circumstances.

For SQL Anywhere Server, you can view the environment variables set for a particular server by starting the server with the -ze option. See "-ze dbeng12/dbsrv12 server option" on page 243.

Setting environment variables on Windows

The SQL Anywhere installer creates or modifies the following environment variables in your computer's properties: PATH and SQLANY12. After installing SQL Anywhere, you must restart your computer for these environment variables to take effect.

Other environment variables can be set by modifying the properties for your computer, or within command prompts or batch files by using the SET command.

Setting environment variables on Unix and Mac OS X

Once SQL Anywhere is installed, each user must set some environment variables for the system to locate and run SQL Anywhere applications. The SQL Anywhere installer creates two files, *sa_config.sh* and *sa_config.csh*, for this purpose. These files are installed in *install-dir/bin32* and *install-dir/bin64*. Each file sets all needed user environment variables.

As the names imply, one file is designed to work under Bourne shell (*sh*) and its derivatives (such as *ksh* or *bash*). The other file is designed to work under C-shell (*csh*) and its derivatives (such as *tcsh*).

Some statements are commented out in each of these batch files. The system administrator may want to edit these files and remove comments, depending on the configuration of their system.

To run a SQL Anywhere application, you have several choices:

1. If you add the environment variables from the *sa_config* files to your system environment, you can run applications by launching them from a GUI, such as X window server, or by typing the application name in a terminal window.

2. In a terminal window, if you source one of the *sa_config* files, you can run the application by typing its name. See "Sourcing files on Unix and Mac OS X" on page 378.

3. *install-dir/bin32s* and *install-dir/bin64s* contain scripts with the same names as SQL Anywhere applications. These scripts set the appropriate environment variables before launching the application. You can run the application by running the corresponding script. You do not have to source an *sa_config* file before you run these scripts.

Setting environment variables for the Finder on Mac OS X

The SQL Anywhere installer sets the following environment variables: DYLD_LIBRARY_PATH, ODBCINI, PATH, and SQLANY12. Rebooting is not required.

Terminal sessions do not inherit environment variables from the Finder. For information about setting environment variables for terminal sessions, see "Sourcing files on Unix and Mac OS X" on page 378.

Sourcing files on Unix and Mac OS X

To **source** a file means to execute commands contained in a text file in the current instance of the shell. This is accomplished using a command built into the shell.

Under Bourne shell and its derivatives, the name of this command is `.` (a single period). For example, if SQL Anywhere is installed in */opt/sqlanywhere12*, the following statement sources *sa_config.sh*:

```
.  /opt/sqlanywhere12/bin32/sa_config.sh
```

For example, on a Mac, run the following statement to source *sa_config.sh*:

```
. Applications/SQLAnywhere12/System/bin32/sa_config.sh
```

Under C-shell and its derivatives, the command is `source`. For example, if SQL Anywhere is installed in */opt/sqlanywhere12*, the following statement sources *sa_config.csh*:

```
source  /opt/sqlanywhere12/bin32/sa_config.csh
```

Configuring the samples

Before using the samples for SQL Anywhere 12, run the configuration script corresponding to the bitness of SQL Anywhere you want to use with them. For example, to configure the samples for use with 32-bit software, run:

```
. /opt/sqlanywhere12/samples/sample_config32.sh
```

For example, on a Mac, to configure the samples for use with a 32-bit software, run:

```
. Applications/SQLAnywhere12/samples/sample_config32.sh
```

DYLD_LIBRARY_PATH environment variable [Mac OS X]

Specifies the directories that are searched at run time for libraries required by SQL Anywhere applications on Mac OS X.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

DYLD_LIBRARY_PATH=*path-list*

Default

/Applications/SQLAnywhere12/System/lib32

Remarks

The *sa_config.sh* and *sa_config.csh* files, created by the installer, are scripts that create or modify this and other environment variables.

See also

- "LD_LIBRARY_PATH environment variable [Linux and Solaris]" on page 379
- "LIBPATH environment variable [IBM AIX]" on page 379
- "SHLIB_PATH environment variable [HP-UX]" on page 387
- "Setting environment variables on Unix and Mac OS X" on page 377

LD_LIBRARY_PATH environment variable [Linux and Solaris]

Specifies the directories that are searched at run time for libraries required by SQL Anywhere applications on Linux and Solaris.

Syntax

LD_LIBRARY_PATH=*path-list*

Default

- */opt/sqlanywhere12/lib32* (32-bit platforms)
- */opt/sqlanywhere12/lib64* (64-bit platforms)

Remarks

The *sa_config.sh* and *sa_config.csh* files, created by the installer, are scripts that create or modify this and other environment variables.

See also

- "DYLD_LIBRARY_PATH environment variable [Mac OS X]" on page 378
- "LIBPATH environment variable [IBM AIX]" on page 379
- "SHLIB_PATH environment variable [HP-UX]" on page 387
- "Setting environment variables on Unix and Mac OS X" on page 377

LIBPATH environment variable [IBM AIX]

Specifies the directories that are searched at run time for libraries required by SQL Anywhere applications on IBM AIX.

Syntax

LIBPATH=*path-list*

Default

- */usr/lpp/sqlanywhere12/lib32* (32-bit platforms)
- */usr/lpp/sqlanywhere12/lib64* (64-bit platforms)

Remarks

The *sa_config.sh* and *sa_config.csh* files, created by the installer, are scripts that create or modify this and other environment variables.

See also

- "DYLD_LIBRARY_PATH environment variable [Mac OS X]" on page 378
- "LD_LIBRARY_PATH environment variable [Linux and Solaris]" on page 379
- "SHLIB_PATH environment variable [HP-UX]" on page 387
- "Setting environment variables on Unix and Mac OS X" on page 377

ODBCHOME environment variable [Unix]

Specifies the location of the *.odbc.ini* file.

Syntax

ODBCHOME=*odbc-ini-directory*

Remarks

The *.odbc.ini* file is the system information file that contains ODBC data sources. If the file is named anything other than *.odbc.ini*, you must use the ODBCINI or ODBC_INI environment variable to specify its location.

For information about the algorithm for locating ODBC data sources, see "Using ODBC data sources on Unix" on page 105.

See also

- "ODBCINI and ODBC_INI environment variables [Unix]" on page 380
- "Setting environment variables on Unix and Mac OS X" on page 377

ODBCINI and ODBC_INI environment variables [Unix]

Specifies the path and name of the system information file containing ODBC data sources.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

ODBCINI=*odbc-ini-file*

ODBC_INI=*odbc-ini-file*

Remarks

The file name does not need to be *.odbc.ini* if it is specified using one of these environment variables. Both environment variables are provided for compatibility with other products.

For information about the algorithm for locating ODBC data sources, see "Using ODBC data sources on Unix" on page 105.

See also

- "ODBCHOME environment variable [Unix]" on page 380
- "Setting environment variables on Unix and Mac OS X" on page 377

PATH environment variable

Specifies the locations of directories containing SQL Anywhere executables.

Syntax

PATH=*path-list*

Default

The following paths are only added if the corresponding component is installed.

Operating system	Default location
Windows (32-bit)	*C:\Program Files\SQL Anywhere 12\Bin32*
Windows (64-bit)	*C:\Program Files\SQL Anywhere 12\bin64*
Mac OS X (32-bit)	*/Applications/SQLAnywhere12/System/bin32*
Mac OS X (64-bit)	*/Applications/SQLAnywhere12/System/bin64*
IBM AIX (32-bit)	*/usr/lpp/sqlanywhere12/bin32*
IBM AIX (64-bit)	*/usr/lpp/sqlanywhere12/bin64*
Other Unix operating systems (32-bit)	*/opt/sqlanywhere12/bin32*
Other Unix operating systems (64-bit)	*/opt/sqlanywhere12/bin64*
Linux, Solaris Sparc	*/opt/sqlanywhere12/openserver/OCS-15_0/bin*

Remarks

On Windows, the PATH environment variable is modified by the installer to include the directories where SQL Anywhere executables are located.

On Unix, the *sa_config.sh* and *sa_config.csh* files, created by the installer, are scripts that create or alter this and other environment variables.

See also

- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377
- "Sourcing files on Unix and Mac OS X" on page 378

SACHARSET environment variable

Specifies the character set used by SQL Anywhere.

Syntax

SACHARSET=*charset-label*

Remarks

The *charset-label* is a character set label.

For information about the recommended character set labels for Windows and Unix operating systems, see "Recommended character sets and collations" on page 436.

You can use the dbinit -le option to obtain a list of all of the available character set labels for a SQL Anywhere database. The dbinit -l option returns a list of available collations for a SQL Anywhere database. See "Initialization utility (dbinit)" on page 799.

If SACHARSET is not specified, the character set comes from the operating system.

See also

- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377

SADIAGDIR environment variable

Specifies the location of the SQL Anywhere diagnostic directory.

Syntax

SADIAGDIR=*diagnostic-information-directory*

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Default

Operating system	Default location
Windows	*%ALLUSERSPROFILE%\Application Data\SQL Anywhere 12\diagnostics*
Unix	*$HOME/.sqlanywhere12/diagnostics*
Windows Mobile	Directory where the database server is running

Remarks

SQL Anywhere stores crash reports and feature statistics information in a diagnostic directory. The SADIAGDIR environment variable is used to determine the location of the diagnostic directory where SQL Anywhere writes crash reports.

If the directory specified by this environment variable does not exist, then the database server operates as though the environment variable is not set.

On Windows (except Windows Mobile), diagnostics are written to the first writable directory in the following list:

1. The directory specified by the SADIAGDIR environment variable.
2. The directory of the current executable.
3. The current directory.
4. The temporary directory. See "SATMP environment variable" on page 385 and "TMP, TMPDIR, and TEMP environment variables" on page 390.

On Windows Mobile, diagnostics are written to the first writable directory in the following list:

1. The directory of the current executable.
2. The current directory.
3. The temporary directory. See "Registry settings on Windows Mobile" on page 399.

On Unix, diagnostics are written to the first writable directory in the following list:

1. The directory specified by the SADIAGDIR environment variable.
2. The directory specified by *$HOME/.sqlanywhere12/diagnostics*.
3. The current directory.
4. The temporary directory. See "SATMP environment variable" on page 385 and "TMP, TMPDIR, and TEMP environment variables" on page 390.

Note
On Unix, writing crash reports to the user's home directory is not recommended when the database or MobiLink server is running as a daemon, or the user is root/nobody. Because of this, the Unix install prompts you for a suitable location and sets the SADIAGDIR environment variable in the *sa_config.sh* and *sa_config.csh* files.

See also

- "Support utility (dbsupport)" on page 853
- "Error reporting in SQL Anywhere" on page 996
- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377

SALANG environment variable

Specifies the language code for SQL Anywhere.

Syntax

SALANG=*language-code*

Remarks

The *language-code* is a two-letter combination representing a language. For example, setting **SALANG=DE** sets the default language to German.

For information about supported language codes, see "Understanding the locale language" on page 417.

The first of the following methods that returns a value determines the default language:

1. Check the SALANG environment variable.
2. (Windows) Check the registry as set during installation or by *dblang.exe*. See "Language Selection utility (dblang)" on page 818.
3. Query the operating system for language information.
4. If no language information is set, English is the default.

See also

- "Language Selection utility (dblang)" on page 818
- "Registry settings on installation" on page 398
- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377

SALOGDIR environment variable

Specifies the location of the *backup.syb* file.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

SALOGDIR=*directory-name*

Remarks

If the SALOGDIR environment variable is set, it is assumed to contain the path for a directory where the backup history file, *backup.syb* can be written. This file is updated each time you execute a BACKUP or RESTORE statement.

On Windows, the *backup.syb* file is written in the first writable location in the following list:

1. The SALOGDIR environment variable.
2. *%ALLUSERSPROFILE%\SQL Anywhere 12*.
3. The root directory of the current drive.
4. The current directory.

On Windows CE, the *backup.syb* file is written in the first writable location in the following list:

1. The SALOGDIR environment variable.
2. The directory where the database server was started.
3. The root directory.
4. The current directory.

On Unix, the *backup.syb* file is written in the first writable location in the following list:

1. The SALOGDIR environment variable.
2. *$HOME/.sqlanywhere12*.
3. The current directory.

See also

- "BACKUP statement" [*SQL Anywhere Server - SQL Reference*]
- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377

SATMP environment variable

Specifies the location of temporary files used by the database server and the SQL Anywhere command line utilities that require a temporary directory.

Syntax

SATMP=*directory-name*

Remarks

SQL Anywhere creates two types of temporary files: database server-related temporary files (created on all platforms), and communications-related temporary files (created only on Unix for both the client and the server).

The SATMP environment variable specifies the location of temporary files used by the database server and the SQL Anywhere command line utilities that require a temporary directory. It is useful when running the database server as a service because it enables you to hold the temporary file in a directory that cannot be accessed by other programs.

If the location of the temporary file is not specified with the -dt option when the database server is started, then the database server checks the value of the SATMP environment variable to determine where to place the temporary file. If the SATMP environment variable does not exist, then the first of the TMP, TMPDIR, or TEMP environment variables to exist is used. On Unix, if none of the above environment variables exist, */tmp* is used.

On Windows Mobile, you can specify the directory to use as the server's temporary directory in the registry.

For information about the temporary file location on Windows Mobile, see "Registry settings on Windows Mobile" on page 399.

On Unix, both the client and the database server must set SATMP to the same value when connecting via shared memory.

For information about securing shared memory connections on Unix, see "Security tips" on page 1116.

Using shared memory connections with older software
In SQL Anywhere version 9 and earlier, the environment variable ASTMP is equivalent to SATMP. If you are using shared memory to connect version 9 and version 10 software, you must set the SATMP and ASTMP environment variables to specify the (same) location of the temporary directory.

If you want to restrict the permissions of the temporary files created by the database server or client on Unix, you must set this environment variable to a directory that is not in the following list:

- */tmp*
- */tmp/.SQLAnywhere*
- the value of the TMP environment variable, if set
- the value of the TMPDIR environment variable, if set
- the value of the TEMP environment variable, if set
- a symbolic link pointing to any of the above directories

When SATMP is set to a directory that is not listed above, the database server searches up the given directory path looking for directories owned by the current user with permissions set to 770, 707, or 700. If the permissions are not set to one of these values, files are created with permissions set to 777. For each directory that is found, the database server removes the appropriate permissions (other, group, and other +group, respectively) from the permission mask used to create temporary files.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Caution
Setting SATMP to a directory that is not included in the list above may affect the ability of users using different Unix accounts to connect to the database server over shared memory.

See also

- "-dt dbeng12/dbsrv12 server option" on page 175
- "TMP, TMPDIR, and TEMP environment variables" on page 390
- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377
- "Place different files on different devices" [*SQL Anywhere Server - SQL Usage*]

SHLIB_PATH environment variable [HP-UX]

Specifies the directories that are searched at run time for libraries required by SQL Anywhere applications on HP-UX.

Syntax

SHLIB_PATH=*path-list*

Default

- */opt/sqlanywhere12/lib32* (32-bit platforms)
- */opt/sqlanywhere12/lib64* (64-bit platforms)

Remarks

The *sa_config.sh* and *sa_config.csh* files, created by the installer, are scripts that create or modify this and other environment variables.

See also

- "DYLD_LIBRARY_PATH environment variable [Mac OS X]" on page 378
- "LD_LIBRARY_PATH environment variable [Linux and Solaris]" on page 379
- "LIBPATH environment variable [IBM AIX]" on page 379
- "Setting environment variables on Unix and Mac OS X" on page 377

SQLANY12 environment variable

Specifies the location of the directory containing SQL Anywhere 12.

Syntax

SQLANY12=*directory-name*

Default

Operating system	Location
Windows	*C:\Program Files\SQL Anywhere 12*
IBM AIX	*/usr/lpp/sqlanywhere12*
Mac OS X	*/Applications/SQLAnywhere12/System*
Other Unix operating systems	*/opt/sqlanywhere12*

Remarks

This environment variable should be set for several reasons. For example, samples require this environment variable to locate SQL Anywhere applications.

On Windows, the installer sets the location of the SQLANY12 environment variable.

On Unix, the *sa_config.sh* and *sa_config.csh* files, created by the installer, are scripts that create or modify this and other environment variables.

See also

- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377

SQLANYSAMP12 environment variable

Specifies the location of the SQL Anywhere samples directory.

Syntax

SQLANYSAMP12=*directory-name*

Default

Operating system	Default location
Windows (except Windows Mobile, Windows Vista, Windows 7, and Windows 2008)	*C:\Documents and Settings\All Users\Documents\SQL Anywhere 12\Samples*
Windows Vista, Windows 7, Windows 2008	*C:\Users\Public\Documents\SQL Anywhere 12\Samples*

Remarks

On Windows, the installer sets the location of the SQLANYSAMP12 environment variable.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Samples directory" on page 392
- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377

SQLCONNECT environment variable

Specifies additional connection parameters used when connecting to the database server.

Syntax

SQLCONNECT=*parameter=value*; ...

Remarks

This string is a list of parameter settings, of the form *parameter=value*, delimited by semicolons.

Connection parameters specified by the SQLCONNECT environment variable are not used if they have already been specified in the connection string.

For information about the supported connection parameters, see "Connection parameters" on page 265.

Password security risk
Because the password is in plain text, putting it into the SQLCONNECT environment variable is a security risk.

See also

- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377

SQLPATH environment variable

Specifies the location of command and Help files.

Syntax

SQLPATH=*path-list*

Remarks

Interactive SQL searches the directories specified in SQLPATH for command files and Help files before searching the system path.

See also

- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377

SQLREMOTE environment variable

Specifies subdirectories that are addresses for the SQL Remote FILE message link.

Syntax

SQLREMOTE=*path*

Remarks

Addresses for the FILE message link in SQL Remote are subdirectories of the SQLREMOTE environment variable. This environment variable should specify a shared directory.

On Windows operating systems, except Windows Mobile, an alternative to setting the SQLREMOTE environment variable is to set the *SQL Remote\Directory* registry entry to the proper root directory.

See also

- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377

SYBASE environment variable

Specifies the home directory for the installation of some Sybase applications, including Adaptive Server Enterprise, Sybase Open Client, Sybase Open Server, and utilities such as DSEdit.

Syntax

SYBASE=*directory-name*

Remarks

You only need to set this environment variable if you are using other Sybase applications.

See also

- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377

TMP, TMPDIR, and TEMP environment variables

Specifies the location of SQL Anywhere temporary files.

Syntax

TMP=*path*

TMPDIR=*path*

TEMP=*path*

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

SQL Anywhere software may create temporary files for various operations. A temporary file is created when the database server starts, and is erased when the database server stops. As its name suggests, the temporary file is used while the database server is running to hold temporary information. The temporary file does not hold information that needs to be kept between sessions.

Temporary files are held in the directory specified by one of the TMP, TMPDIR, or TEMP environment variables. If more than one of these environment variables is specified, then the first of TMP, TMPDIR, and TEMP is used.

SQL Anywhere Server checks the SATMP environment variable first. If it is not specified, then these environment variables are checked. See "SATMP environment variable" on page 385.

If none of the environment variables is defined, temporary files are placed in the current working directory of the server. On Unix only, if none of these environment variables are found, then */tmp* is used.

On Windows Mobile, you can use the registry to specify the directory to use as the server's temporary directory.

For more information about setting the temporary directory value, see "Registry settings on Windows Mobile" on page 399.

Using shared memory connections with older software
In SQL Anywhere version 9 and earlier, the environment variable ASTMP is equivalent to SATMP. If you are using shared memory to connect version 9 and version 10 software, you must set the SATMP and ASTMP environment variables to specify the location of the temporary file.

See also

- "-dt dbeng12/dbsrv12 server option" on page 175
- "SATMP environment variable" on page 385
- "Setting environment variables on Windows" on page 377
- "Setting environment variables on Unix and Mac OS X" on page 377
- "Place different files on different devices" [*SQL Anywhere Server - SQL Usage*]

File locations and installation settings

When you install SQL Anywhere, several directories are created. Some of the files in these directories are essential, and others are not. This section describes the directory structure.

SQL Anywhere software, whether you receive it as a product or bundled as part of another product, is installed under a single installation directory. The SQLANY12 environment variable specifies the location of the installation directory. See "SQLANY12 environment variable" on page 387.

SQL Anywhere installation directory

The SQL Anywhere installation directory itself holds several items, including the following:

- **Read Me First** A Read Me First file named *readme.txt* holds last minute information.

For platforms other than Windows Mobile, there are several directories under the installation directory:

- **Executable directories** There is a separate directory for each operating system platform, which holds configuration files and context-sensitive help files.

  On Windows, except Windows Mobile, these files are installed in the *bin32* or *bin64* directory. If you are using Unix, they are installed in the *bin32* or *bin64* and *lib32* or *lib64* directories.

  You only have the directories required for your operating system version.

- ***java* directory** JAR files are stored in this directory.

- ***scripts* directory** The scripts directory contains SQL scripts that are used by the database administration utilities and as examples.

- ***\SDK\Include* directory** The *\SDK\Include* directory contains header files for developing C/C++ applications for SQL Anywhere. On Unix, this directory is called *include*.

Windows Mobile file locations

On Windows Mobile devices, all files are installed in the installation directory *\Program Files\SQLAny12*, except for DLLs, which are installed in the *\Windows* directory. No subdirectories are created.

Unix file locations

The language resources are installed in the *res* directory, and the shared objects are installed in the *lib32* or *lib64* directory.

Samples directory

When you install SQL Anywhere, you can choose the directory where the samples are installed. The documentation refers to this location as *samples-dir*.

The SQLANYSAMP12 environment variable specifies the location of *samples-dir*. See "SQLANYSAMP12 environment variable" on page 388.

On Windows, you can access the samples from the **Start** menu by choosing **Programs** » **SQL Anywhere 12** » **Sample Applications And Projects**.

The following table shows default and typical locations of *samples-dir* for each supported operating system:

Operating system	Default installation location (*samples-dir*)	Typical installation location
Windows XP/200x	*%ALLUSERSPROFILE%\Documents \SQL Anywhere 12\Samples*	*C:\Documents and Settings\All Users \Documents\SQL Anywhere 12\Samples* [1]

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Operating system	Default installation location (*samples-dir*)	Typical installation location
Windows Vista and Windows 7	*%PUBLIC%\Documents\SQL Anywhere 12\Samples*	*C:\Users\Public\Documents\SQL Anywhere 12\Samples*
Windows Mobile	*\Program Files\SQLAny12*	
Mac OS X	*/Applications/SQLAnywhere12/Samples*	
IBM AIX	*/usr/lpp/sqlanywhere12/samples*	
Other Unix operating systems	*/opt/sqlanywhere12/samples*	

[1] When accessing the SQL Anywhere samples directory in Windows Explorer, the location is *Documents and Settings* > *All Users* > ***Shared Documents*** > *SQL Anywhere 12* > *Samples*. However, if you are accessing the SQL Anywhere samples directory from a command prompt, the path is *C:\Documents and Settings\All Users**Documents***\SQL Anywhere 12\Samples*.

How SQL Anywhere locates files

The client library and the database server need to locate files for two main purposes:

- DLLs and initialization files are required to run SQL Anywhere. If an incorrect DLL is located, there is the possibility of version mismatch errors.
- Some files are specified in SQL statements and need to be located at run time, such as INSTALL JAVA or LOAD TABLE.

Examples of SQL statements that use file names include the following:

- **INSTALL JAVA statement** The name of the file that holds Java classes.
- **LOAD TABLE and UNLOAD TABLE statements** The name of the file from which data should be loaded or to which the data should be unloaded.
- **CREATE DATABASE statement** A file name is needed for this statement and similar statements that can create files.

Sometimes SQL Anywhere uses a simple algorithm to locate files. In other cases, a more extensive search is performed.

Simple file searching

In many SQL statements (such as LOAD TABLE, or CREATE DATABASE), the file name is interpreted as relative to the current working directory of the database server.

Also, when a database server is started and a database file name (DatabaseFile (DBF) parameter) is supplied, the path is interpreted as relative to the current working directory.

Extensive file searching on Windows

On Windows, SQL Anywhere programs, including the database server and administration utilities, can perform a more extensive search for required files such as DLLs or shared libraries. In these cases, SQL Anywhere programs look for files in the following order:

1. The module's directory (the directory where the program executable file or library file is located).
2. The executable directory (the directory where the program executable file or library is located).
3. The installation path (the SQL Anywhere installation directory, *install-dir*). *install-dir* is a single directory specified by the SQLANY12 environment variable if it is defined.
4. No path (the current working directory).
5. *%APPDATA%\SQL Anywhere 12* directory
6. *%ALLUSERSPROFILE%\SQL Anywhere 12* directory
7. The Location registry entry.
8. System-specific directories. This includes directories where common operating system files are held, such as the *Windows* directory and the *Windows\system32* directory on Windows operating systems.
9. The PATH directories. Directories in the system path and the user's path are searched.

Note
On Windows, SQL Anywhere searches the following paths relative to each location in the preceding list:

1. .
2. ..
3. .*bin32* and ..*bin32* (32-bit programs only)
4. .*bin64* and ..*bin64* (64-bit programs only)
5. .*java* (for Java-related files)
6. ..*java* (for Java-related files)
7. .*scripts* (for SQL script files)
8. ..*scripts* (for SQL script files)

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Extensive file searching on Windows Mobile

On Windows Mobile, SQL Anywhere programs, including the database server and administration utilities, can perform a more extensive search for required files such as DLLs or shared libraries. In these cases, SQL Anywhere programs look for files in the following order:

1. The module's directory (the directory where the program executable file or library file is located).
2. The executable directory (the directory where the program executable file or library is located).
3. No path (the current working directory).
4. The Location registry entry.
5. System-specific directories. This includes directories where common operating system files are held, such as *Windows*.

Note
On Windows Mobile, SQL Anywhere searches the following paths relative to each location in the preceding list:

1. .
2. ..
3. .*bin32*
4. ..*bin32*
5. .*java* (for Java-related files)
6. ..*java* (for Java-related files)
7. .*scripts* (for SQL script files)
8. ..*scripts* (for SQL script files)

Extensive file searching on Unix

On Unix, SQL Anywhere programs, including the database server and administration utilities, can perform a more extensive search for required files such as DLLs or shared libraries. In these cases, SQL Anywhere programs look for files in the following order:

1. The executable path (if it can be determined).
2. The installation path (the SQL Anywhere installation directory, *install-dir*). *install-dir* is a single directory specified by the SQLANY12 environment variable if it is defined.

3. No path (the current working directory).
4. *$HOME/.sqlanywhere12* directory
5. The PATH environment variable.
6. The LIBPATH environment variable:
   - LD_LIBRARY_PATH on Linux and Solaris
   - LD_LIBRARY_PATH and SHLIB_PATH on HP-UX
   - LIBPATH on IBM AIX
   - DYLD_LIBRARY_PATH on Mac OS X

Note
On Unix, SQL Anywhere searches the following paths relative to each location in the preceding list:

1. .
2. ..
3. *./bin32* and *../bin32* (32-bit programs only)
4. *./bin64* and *../bin64* (64-bit programs only)
5. *./lib32* and *../lib32* (library files for 32-bit programs only)
6. *./lib64* and *../lib64* (library files for 64-bit programs only)
7. *./java* (for Java-related files)
8. *../java* (for Java-related files)
9. *./scripts* (for SQL script files)
10. *../scripts* (for SQL script files)
11. *./res* (for *.res* files)
12. *../res* (for *.res* files)
13. *./tix* (for *.tix* files)
14. *../tix* (for *.tix* files)

Registry and INI files

On Windows operating systems (except Windows Mobile), SQL Anywhere uses several registry settings. On Unix, these settings are stored in initialization files instead.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The software installation makes these settings for you, and in general operation you should not need to access the registry or initialization files. The settings are provided here for those people who make modifications to their operating environment.

The contents of *.ini* files used by SQL Anywhere can be obfuscated with simple encryption using the File Hiding utility. See "File Hiding utility (dbfhide)" on page 794.

Caution
You should not add simple encryption to the system information file (named *.odbc.ini* by default) with the File Hiding utility (dbfhide) on Unix unless you are only using SQL Anywhere data sources. If you plan to use other data sources (for example, for MobiLink synchronization), then obfuscating the contents of the system information file may prevent other drivers from functioning properly.

Current user and local machine settings

Some operating systems hold two levels of system settings. Some settings are specific to an individual user and are used only when that user is logged on; these settings are called **current user** settings. Some settings are global to the computer, and are available to all users; these are called **local machine** settings. You must have administrator permissions on your computer to change local machine settings.

SQL Anywhere respects both current user and local machine settings. On Windows XP, for example, these are held in the *HKEY_CURRENT_USER* key and the *HKEY_LOCAL_MACHINE* key, respectively.

Current user takes precedence

If a setting is made in both the current user and local machine registries, the current user setting takes precedence over the local machine setting.

When local machine settings are needed

If you are running a SQL Anywhere program as a **service**, you should ensure that the settings are made at the local machine level.

Services can continue to run under a special account when you log off a computer as long as you do not shut the computer down entirely. They can be made independent of individual accounts, and therefore need access to local machine settings.

In addition to SQL Anywhere programs, some web servers run as services. You must set local machine settings for Apache or Microsoft IIS to work with such a web server.

In general, the use of local machine settings is recommended.

Hiding the contents of .ini files

Often, SQL Anywhere expects an *.ini* file to have a particular name. When you want to add simple encryption to a file whose name is important (such as *saldap.ini*), you must save a copy of the original file with a different name when you add simple encryption to the file. If you do not keep a copy of the original

file, then you cannot modify the contents of the file once it has been obfuscated. The following steps explain how to add simple encryption to a *.ini* file.

To hide the contents of a file

1. Save the file with a different name.

   ```
   rename saldap.ini saldap.ini.org
   ```

2. Obfuscate the file with the File Hiding utility, giving the obfuscated file the required file name.

   ```
   dbfhide saldap.ini.org saldap.ini
   ```

3. Protect the *saldap.ini.org* file using file system or operating system protection, or store the file in a secure location.

   To make a change to the *saldap.ini* file, edit the *saldap.ini.org* file and repeat step 2.

Caution
You should not add simple encryption to the system information file (named *.odbc.ini* by default) with the File Hiding utility (dbfhide) on Unix unless you will only be using SQL Anywhere data sources. If you plan to use other data sources (for example, for MobiLink synchronization), then obfuscating the contents of the system information file may prevent other drivers from functioning properly.

See also

- "File Hiding utility (dbfhide)" on page 794

Registry structure

On Windows (except Windows Mobile), you can access the registry directly with the registry editor. The SQL Anywhere registry entries are held in either the *HKEY_CURRENT_USER* or *HKEY_LOCAL_MACHINE* keys, in the following location:

```
Software
   Sybase
      SQL Anywhere
         12.0
      Sybase Central
         6.1.0
```

Caution
Modify your registry at your own risk. It is recommended that you back up your system before modifying the registry.

Registry settings on installation

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Caution
To improve robustness across power failures on systems using certain Intel storage drivers, specific registry entries must be set. Failure to set this parameter can result in lost data and corrupted databases in the event of a power failure. See "To improve robustness on Intel storage drivers" [*SQL Anywhere Server - Programming*].

On Windows, the installation program makes the following settings in the *HKEY_LOCAL_MACHINE \Software\Sybase* registry. The following list describes some of these registry settings:

- **SQL Anywhere\12.0\Location** This entry holds the installation directory location for the SQL Anywhere software. For example:

  ```
  Location "C:\Program Files\SQL Anywhere 12"
  ```

- **SQL Anywhere\12.0\Samples Location** This entry holds the installation directory location for sample programs. For example:

  ```
  Samples Location "C:\Documents and Settings\All Users\Documents\SQL
  Anywhere 12\Samples\"
  ```

- **SQL Anywhere\12.0\Online Resources** This entry holds the location for the Online Resources documentation. For example:

  ```
  Online Resources "C:\Program Files\SQL Anywhere 12\support\ianywhere.html"
  ```

- **SQL Anywhere\12.0\Language** This entry holds a two-letter code indicating the current language for messages and errors. For example:

  ```
  Language "EN"
  ```

  The language is set based on the language selection specified during installation. See "Understanding the locale language" on page 417.

- **Sybase Central\6.1.0\Language** This entry holds a two-letter code indicating the current language for messages and errors. For example:

  ```
  Language "EN"
  ```

  This entry is used by Sybase Central. The language is set based on the language selection specified during installation. See "Understanding the locale language" on page 417.

Registry settings on Windows Mobile

You can specify which directory you want to use as the server's temporary directory on Windows Mobile by setting the following value in the registry:

HKEY_CURRENT_USER\Software\Sybase\SQL Anywhere\12.0\TempFolder

TempFolder is the name of the temporary directory you want to use. The server does one of the following:

- use the specified directory if it exists.
- attempt to create the specified directory if it does not already exist, as long as the parent directory already exists.

If the specified directory does not exist and cannot be created, the database server:

- uses the *\Temp* directory if it exists.
- attempts to create a *\Temp* directory if it does not already exist.

If the *\Temp* directory does not exist and cannot be created, the server uses the current directory.

International languages and character sets

Internationalization refers to the ability of software to handle a variety of languages and their appropriate characters sets, independently of the language in which the software is running, or the operating system on which the software is running. SQL Anywhere has full internationalization capabilities. The following features discuss the most commonly requested and used capabilities.

- **Unicode support** SQL Anywhere supports Unicode as follows:
  - Client support for UTF-16 in SQL Anywhere client libraries for ODBC, OLE DB, ADO.NET, and JDBC
  - NCHAR data types for storing Unicode character data in UTF-8
  - CHAR data types can use UTF-8 encoding
- **Code pages and character sets** The SQL Anywhere database server and related tools support Windows (ANSI/ISO), UTF-8, and Unix code pages and character sets.
- **Collations** SQL Anywhere supports two collation algorithms: the SQL Anywhere Collation Algorithm (SACA), and the Unicode Collation Algorithm (UCA) using International Components for Unicode (ICU).

  For more information about ICU, see "What is ICU, and when is it needed?" on page 403.

  SACA provides fast, compact, and reasonable sorting at the expense of linguistic correctness. UCA provides linguistic correctness, but with a small expense in storage requirements and execution time. See "Understanding collations" on page 419.

  For advanced ordering and comparison capabilities, SQL Anywhere also provides the SORTKEY and COMPARE functions. These functions provide advanced linguistic sorting capabilities, like the ordering found in a dictionary or telephone book. Where appropriate, case-insensitive and accent-insensitive ordering and comparisons are provided. See "SORTKEY function [String]" [*SQL Anywhere Server - SQL Reference*] and "COMPARE function [String]" [*SQL Anywhere Server - SQL Reference*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

SQL Anywhere also contains design features allowing for automatic use of SORTKEY-based ordering on character columns. The sort_collation database option specifies the sort ordering to be used when an ORDER BY is specified for a character column. Computed columns may also be used to store sort keys for character columns so that they do not need to be computed each time that an ORDER BY is specified. See "sort_collation option" on page 588.

- **Character set conversion** SQL Anywhere converts data between the character set encoding on your server and client systems, and maintains the integrity of your data, even in mixed character set environments. See "Character set conversion" on page 410.

- **Identifiers** SQL Anywhere supports the use of identifiers containing most single-byte and multibyte characters without requiring quotes. Exceptions are generally limited to spaces and punctuation symbols.

- **Currency** Currency symbols, including the euro symbol, are supported for ordering. SQL Anywhere provides no currency formatting support.

- **Date and time formats** SQL Anywhere supports the Gregorian calendar, and provides a variety of formats for date and time strings. Custom formatting can be done using the date_format, time_format, and timestamp_format database options. The date_format and timestamp_format options default to an ISO-compatible format for the date, YYYY-MM-DD. SQL Anywhere provides the CONVERT function, which provides output formatting of dates and times into a variety of popular formats. See:

  - "date_format option" on page 528
  - "time_format option" on page 603
  - "timestamp_format option" on page 604
  - "CONVERT function [Data type conversion]" [*SQL Anywhere Server - SQL Reference*]

See also

- "Create a database with a named collation" on page 433
- "Recommended character sets and collations" on page 436

Localized versions of SQL Anywhere

Localization refers to the linguistic and cultural adaptation of a product to a target locale, which is usually a combination of language and country/region. Localization affects many components, including packaging, installation, documentation, software user interface, and error/warning/information messages.

SQL Anywhere software is localized to five languages:

- English
- French
- German
- Japanese
- Simplified Chinese

Language choice is determined at installation.

Localized versions of the documentation are available in English, German, Japanese, and Simplified Chinese.

On Windows, **Start** menu items allow the software to be reconfigured between the installed language and English. The Language Selection utility (dblang) allows the software to be reconfigured to any of the available languages, including the additional deployment languages. See:

- "Deployment software localization on Windows" on page 403
- "Language Selection utility (dblang)" on page 818
- "Changing administration tool language settings" on page 758

The following table shows the availability of each language by operating system platform.

Platform	English	French	German	Japanese	Simplified Chinese
Windows	Yes	Yes	Yes	Yes	Yes
Windows Mobile	Yes	Yes	Yes	Yes	Yes
Linux	Yes		Yes	Yes	Yes
Unix	Yes				
Mac OS X	Yes				

Full software and documentation localization

SQL Anywhere for Windows is available in the following languages, suitable for development, deployment and administration:

- English
- French
- German
- Japanese
- Simplified Chinese

For English, German, Japanese, and Simplified Chinese, all SQL Anywhere components are localized, including:

- Packaging
- Installer
- Documentation and context-sensitive help
- Software
  - **Start** menu items and program folders

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- Database servers and client libraries
- MobiLink server and client
- SQL Remote client
- Administration tools, including Interactive SQL, Sybase Central, and all related plug-ins
- Command-line tools, such as dbinit and dbunload

For French, the installer, software, and context-sensitive help are localized.

The following components are not localized and are only available in English:

- InfoMaker
- Sybase PowerDesigner Physical Data Model

Deployment software localization on Windows

In addition to the five main languages listed previously, SQL Anywhere provides deployment software resources for the following languages:

- Italian
- Korean
- Lithuanian
- Polish
- Portuguese (Brazilian)
- Russian
- Spanish
- Traditional Chinese
- Ukrainian

Deployment localization applies to a subset of software components typically deployed to end users. Packaging, documentation, administration, development, and installation software are not localized. Localized software components include:

- Database servers and client libraries
- MobiLink server and client
- SQL Remote client
- Command-line tools, such as dbinit and dbunload

What is ICU, and when is it needed?

ICU, or International Components for Unicode, is an open source library developed and maintained by IBM. ICU facilitates software internationalization by providing Unicode support. SQL Anywhere implements certain character set conversions and collation operations using ICU.

When is ICU needed on the database server? (all platforms except Windows Mobile)

Ideally, ICU should always be available for use by the database server. The following table specifies when and why ICU is needed:

ICU is needed when...	Notes
UCA is used as the collation for the NCHAR or CHAR character set.	UCA requires ICU.
The database character set is not UTF-8 but is a multi-byte character set.	For password conversion from the database character set to UTF-8 (database passwords are stored in UTF-8, internally).
The client and database character sets are different, and when either of them is multi-byte (including UTF-8). This includes Unicode ODBC, OLE DB, ADO.NET, and SQL Anywhere JDBC applications, regardless of the database character set where at least one of these clients do not have ICU.	Proper conversion to and from a multi-byte character set requires ICU.
The database character set is not UTF-8 and conversion between CHAR and NCHAR values is required.	The database server requires ICU to convert UTF-8 to another character set.
An embedded SQL client uses an NCHAR character set other than UTF-8.	The database server requires ICU to convert UTF-8 to another character set. Note that the default embedded SQL client NCHAR character set is the same as the initial client CHAR character set. This can be changed using the db_change_nchar_charset function. See "db_change_nchar_charset function" [*SQL Anywhere Server - Programming*].
The CSCONVERT or SORTKEY functions are used. The CSCONVERT function is called to convert between character sets that conform to the requirements of the third point above.	Character set conversion o and from a multi-byte character set requires ICU. Sortkey generation for many sortkey labels requires UCA, which, in turn, requires ICU. See "CSCONVERT function [String]" [*SQL Anywhere Server - SQL Reference*] and "SORTKEY function [String]" [*SQL Anywhere Server - SQL Reference*].

When is ICU needed on the database server? (Windows Mobile)

The following table specifies when and why ICU is needed for Windows Mobile:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

ICU is needed when...	Notes
UCA is used as the NCHAR collation or the CHAR collation.	UCA requires ICU.
The SORTKEY function is used.	Sortkey generation for many sortkey labels requires UCA, which, in turn, requires ICU. See "SORTKEY function [String]" [*SQL Anywhere Server - SQL Reference*].
The CHAR character set does not match the OS character set.	Even if the character sets match, using ICU is recommended because it improves character set conversion if you are using NCHAR, or if the CHAR character set is multibyte.

Note
If you do not install the ICU library, you must choose either a collation whose character set matches the Windows Mobile character set or the UTF8BIN collation as the CHAR collation when creating your database. Also, you must choose the UTF8BIN collation as the NCHAR collation when creating your database.

When can I get correct character set conversion on the database server without ICU?

You can get correct character set conversion without ICU when both the database character set and client character set are single-byte and *sqlany.cvf* is available (all platforms), or if the operating system supports the conversion (Windows only). This is because single-byte to single-byte conversions can be processed without ICU if the *sqlany.cvf* file is available, or the host operating system has the appropriate converters installed.

When is ICU needed on the client? (all platforms except Windows Mobile)

For Unicode client applications, you are likely to get better combined client and database server performance when all clients have ICU installed, regardless of the database character set. This is because some of the required conversion activity may be offloaded from the database server to the client, and because fewer conversions are required.

Also, if you are using ODBC on Windows platforms, you must have ICU installed on the client, even for ANSI applications. This is because the driver manager converts ANSI ODBC calls to Unicode ODBC calls.

Understanding character sets

Each piece of software works with a **character set**. A character set is a set of symbols, including letters, digits, spaces, and other symbols. An example of a character set is ISO-8859-1, also known as Latin1.

To properly represent these characters internally, each piece of software employs an **encoding**, also known as **character encoding**. An encoding is a method by which each character is mapped onto one or more bytes of information, and is presented as a hexadecimal number. An example of an encoding is UTF-8.

Sometimes the terms character set and encoding are used interchangeably, since the two aspects are so closely related.

A **code page** is one form of encoding. A code page is a mapping of characters to numeric representations, typically an integer between 0 and 255. An example of a code page is Windows code page 1252.

In this documentation, the terms encoding, character encoding, character set encoding, and code page are synonymous.

Database servers, which sort characters (for example, listing names alphabetically), use a **collation**. A collation is a combination of a character encoding (a map between characters and their representation) and a **sort order** for the characters. There may be more than one sort order for each character set; for example, a case sensitive order and a case insensitive order, or two languages may sort the same characters in a different order.

Characters are printed or displayed on a screen using a **font**, which is a mapping between characters in the character set and their appearance. Fonts are handled by the operating system.

Operating systems also use a **keyboard mapping** to map keys or key combinations on the keyboard to characters in the character set.

Language issues in client/server computing

Database users working at client applications may see or access strings from the following sources:

- **Data in the database** Strings and other text data are stored in the database. The database server processes these strings when responding to requests. For example, the database server may be asked to supply all the last names beginning with a letter ordered less than N in a table. This request requires string comparisons to be performed, and assumes a character set ordering.
- **Database server software messages** Applications can cause database errors to be generated. For example, an application may submit a query that references a column that does not exist. In this case, the database server returns a warning or error message. This message is held in a **language resource library**, which is a DLL or shared library used by SQL Anywhere.
- **Client application** The client application interface displays text, and internally the client application may process text.
- **Client software messages** The client library uses the same language library as the database server to provide messages to the client application.
- **Operating systems** The client and server operating systems may provide messages or process text.

For a satisfactory working environment, all these sources of text must work together. Loosely speaking, they must all be working in the user's language and/or character set.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Single-byte character sets

Many languages have few enough characters to be represented in a single-byte character set. In such a character set, each character is represented by a single byte: a two-digit hexadecimal number.

At most, 256 characters can be represented in a single byte. No single-byte character set can hold all the characters used internationally, including accented characters. This problem was addressed by the development of a set of code pages, each of which describes a set of characters appropriate for one or more national languages. For example, code page 1253 contains the Greek character set, and code page 1252 contains Western European languages. There are many code pages, and many names for code pages. The above examples are code pages for Windows.

Upper and lower pages

With few exceptions, characters 0 to 127 are the same for all the code pages. The mapping for this range of characters is called the **ASCII** character set. It includes the English language alphabet in upper and lowercase, and common punctuation symbols and the digits. This range is often called the **seven-bit** range (because only seven bits are needed to represent the numbers up to 127) or the **lower** page. The characters from 128 to 255 are called **extended characters**, or **upper** code page characters, and vary from one code page to another.

Problems with code page compatibility are rare if the only characters used are from the English alphabet, as these are represented in the ASCII portion of each code page (0 to 127). However, if other characters are used, as is generally the case in any non-English environment, there can be problems if the database and the application use different code pages.

For example, suppose a database using the UTF-8 character set loads a table from a file containing cp1252 data, and the encoding is not specified as cp1252 on the LOAD TABLE statement. Because the encoding is not specified, the data is assumed to be encoded in UTF-8, so no character conversion takes place; the cp1252 encoding is stored directly in the database. This means that characters such as the euro symbol, represented in cp1252 as hex 80, are not converted into UTF-8. The euro symbol in UTF-8 is represented by the three-byte sequence E2 82 AC, but, in this case, will be stored in the database as 80. Subsequently, when an application requests data, the database server attempts to convert the data from UTF-8 to the client character set. The conversion will produce corrupted characters.

Multibyte character sets

Some languages, such as Japanese and Chinese, have many more than 256 characters. These characters cannot all be represented using a single byte, and therefore must be encoded using a multibyte encoding. In addition, some character sets use the much larger number of characters available in a multibyte representation to represent characters from many languages in a single, more comprehensive, character set. An example of this is UTF-8.

Multibyte character sets may be of **variable width** whereby some characters are single-byte characters; others are double-byte, and so on.

For more information about multibyte character sets and collations, see “SQL Anywhere Collation Algorithm (SACA)” on page 420.

Example

As an example, characters in code page 932 (Japanese) are either one or two bytes in length. If the value of the first byte, also called the **lead byte**, is in the range of hexadecimal values from \x81 to \x9F or from \xE0 to \xFC (decimal values 129-159 or 224-252), the character is a two-byte character and the subsequent byte, also called a **follow byte**, completes the character. A follow byte is any byte(s) other than the first byte.

If the first byte is outside the lead byte range, the character is a single-byte character and the next byte is the first byte of the following character.

ANSI and OEM code pages in Windows

For Windows users, there are two code pages in use. Applications using the Windows graphical user interface use the Windows code page. Windows code pages are compatible with ISO character sets, and also with ANSI character sets. They are often referred to as **ANSI code pages**.

Character-mode applications (those using a command prompt window) in Windows use code pages that were used in DOS. These are called **OEM code pages** (Original Equipment Manufacturer) for historical reasons.

SQL Anywhere supports collations based on both OEM and ANSI code pages. The OEM collations are provided for compatibility, but they should not be used for new databases. See "Supported and alternate collations" on page 423.

Character sets in a SQL Anywhere database

A SQL Anywhere database can use one or two character sets (encodings) for storing character data. The CHAR data types, including CHAR, VARCHAR and LONG VARCHAR, use a single-byte or multibyte character set. UTF-8 may be used. The NCHAR data types, including NCHAR, NVARCHAR, and LONG NVARCHAR, use UTF-8.

SQL statements like LOAD TABLE and functions like CSCONVERT, TO_CHAR, and TO_NCHAR take **db_charset** and **nchar_charset** as parameters to refer to the database CHAR character set and to the database NCHAR character set, respectively.

See also

- "Character data types" [*SQL Anywhere Server - SQL Reference*]
- "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "CSCONVERT function [String]" [*SQL Anywhere Server - SQL Reference*]
- "TO_CHAR function [String]" [*SQL Anywhere Server - SQL Reference*]
- "TO_NCHAR function [String]" [*SQL Anywhere Server - SQL Reference*]

Supported character sets

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

SQL Anywhere supports a growing list of hundreds of character sets and labels. Character set encodings are known by a wide variety of names or labels. To view the list of character sets supported by SQL Anywhere, run the following command:

```
dbinit -le
```

Each line of output lists the most common labels for a given character set encoding, in comma separated form. The first label in each line of output is the preferred SQL Anywhere name for the character set encoding. The others are the labels used by different authorities, organizations, or standards. These are IANA (Internet Assigned Numbers Authority), MIME (Multipurpose Internet Mail Extensions), ICU (International Components for Unicode), Java, and ASE (Adaptive Server Enterprise).

If you do not find the character set you are looking for, you can also execute the following command to see a longer list that includes labels that are less common:

```
dbinit -le+
```

When a character set encoding label is specified, SQL Anywhere searches for the label in the set of labels known to it. Different authorities sometimes use the same label for different character sets. SQL Anywhere does its best to resolve ambiguities by context. For example, a JDBC application that references a character set by an ambiguous label resolves to a Java standard label. It is recommended that the SQL Anywhere label always be used to avoid any ambiguities. For more information about understanding character set encoding labels, see http://site.icu-project.org/.

In addition to the character set encoding labels returned by the dbinit -le option, you can also use the following character set aliases:

- **os_charset** Alias for the character set used by the operating system hosting the database server.
- **char_charset** Alias for the CHAR character set used by the database.
- **nchar_charset** Alias for the NCHAR character set used by the database.

An easy way to determine if a certain character set or label is supported is to test it using the CSCONVERT function. See “CSCONVERT function [String]” [*SQL Anywhere Server - SQL Reference*].

Character set questions and answers

The following table identifies where you can find answers to questions.

To answer the question ...	Consider reading ...
How do I decide which collation to use for my database?	“Understanding collations” on page 419
How are characters represented in software, and in SQL Anywhere in particular?	“Understanding character sets” on page 405

To answer the question ...	Consider reading ...
What collations does SQL Anywhere provide?	"Choosing collations" on page 426
What character set encodings does SQL Anywhere support?	"Supported character sets" on page 408
I have a different character set on client computers from that in use in the database. How can I get characters to be exchanged properly between client and server?	"Character set conversion" on page 410
What character sets can I use for connection strings?	"Connection strings and character sets" on page 410
How do I change the collation sequence of an existing database?	"Change a database from one collation to another" on page 434

Character set conversion

SQL Anywhere can perform character set conversion between character sets that represent the same characters, but at different positions in the character set or code page. There needs to be a degree of compatibility between the character sets for this to be possible. For example, character set conversion is possible between EUC-JIS and cp932 character sets, but not between EUC-JIS and cp1252.

SQL Anywhere implements character set conversion using the International Components for Unicode (ICU) open source library, developed and maintained by IBM.

For more information about using character set conversion to compare values that are in different data types, see "Comparisons between data types" [*SQL Anywhere Server - SQL Reference*].

Connection strings and character sets

If all of your clients do not use the same character sets, connection strings may be a challenge during character set conversion. This is because the connection string is parsed by the client library to locate or start a database server. However, this parsing is done with no knowledge of the character set or language in use by the database server.

The interface library parses the connection string as follows:

1. The connection string is broken down into its *keyword=value* pairs. This can be done independently of the character set, as long as you do not use curly braces {} around CommLinks (LINKS) connection parameters. Instead, use the recommended parentheses (). Curly braces are valid **follow bytes** (bytes other than the first byte) in some multibyte character sets.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

2. The server is located. There is no character set conversion performed on the server name. If the client character set and the database server character set are different, using extended characters in the server name can cause the server to not be found.

   If your clients and servers are running on different operating systems or locales, you should use 7-bit ASCII characters in the server name.

3. The DatabaseName (DBN) or DatabaseFile (DBF) connection parameters are converted from client character set to the database server character set.

4. Once the database is located, the remaining connection parameters are converted to the database's character set.

SQL statements and character sets

SQL Anywhere Server character set conversion causes all SQL statements to be converted to the database character set before parsing and execution. A side-effect of this conversion is that any characters in the SQL statement that cannot be converted to the database character set are converted to a substitution character. A SQL statement with an arbitrary Unicode character can be executed in one of the following ways:

- Use the UNISTR function to specify the Unicode character values
- Use a host variable to specify the Unicode character values
- Use UTF-8 as the database character set

If you select UTF8BIN as the char collation the database character set is UTF-8. If you specify UTF-8 encoding the char collation is UCA.

The Unicode Collation Algorithm (UCA) provides advanced comparison, ordering, and case conversion, but it can affect performance. Although UTF8BIN is space-efficient and fast, the sort order and comparison is binary. Specify the char collation as UTF8BIN if you require Unicode characters in your SQL statements, but do not need the full power of UCA for sorting and comparison. Use UCA only when necessary, by using the SORTKEY and COMPARE functions.

See also

- "SORTKEY function [String]" [*SQL Anywhere Server - SQL Reference*]
- "COMPARE function [String]" [*SQL Anywhere Server - SQL Reference*]
- "Unicode Collation Algorithm (UCA)" on page 420
- "SQL Anywhere Collation Algorithm (SACA)" on page 420
- "Connection parameters passed as connection strings" on page 88

Troubleshooting unexpected symbols when viewing data

When selecting and viewing data using a client application such as Interactive SQL, unexpected symbols such as squares, arrows, and question marks, may appear as characters in the data.

There are two main reasons why this can happen. The first reason is because there is a problem with the underlying data that is stored in the database. For example, if character set conversion was required when the data was inserted into the database, and some characters in the original character set did not have an equivalent character in the database character set, then substitution characters were inserted instead.

The second, and more common, reason why unexpected symbols can appear in the client application is because the font used to display the data does not support the characters. You can resolve this problem by changing to a Unicode font. If it is not possible to change the font for the client application, you can also change the operating system default font.

For example, suppose you are on a Windows system that uses the standard English font (Tahoma), which does not support the display of Japanese characters. However, your database character set is cp932 and the database contains Japanese data, and when you query the database, characters in the results display as small boxes. In Interactive SQL, you can change the font used to display results by choosing **Tools** » **Options** » **Results** » **Font**, and specifying a Unicode font such as Arial Unicode MS, or Lucida Sans Unicode. Unicode fonts are a good choice because they are capable of displaying characters from many languages.

If your client application does not provide font settings that you can change, it is likely using your default operating system font. In this case, consult your operating system documentation for information about how to change the default system font, and change it to a Unicode font.

See also

- "Lossy conversion and substitution characters" [*SQL Anywhere Server - SQL Reference*]
- "Using Interactive SQL" on page 697

International aspects of case sensitivity

SQL Anywhere is always **case preserving** and **case insensitive** for identifiers, such as system view names and column names. The names are stored in the case in which they are created, but any access to the identifiers is done in a case insensitive manner.

For example, the names of the system views are stored in uppercase (SYSDOMAIN, SYSTAB, and so on), but access is case insensitive, so that the two following statements are equivalent:

```
SELECT * FROM systab; SELECT * FROM SYSTAB;
```

The equivalence of upper and lowercase characters is defined in the collation. There are some collations where particular care is required when assuming case insensitivity of identifiers. For example, Turkish collations have a case-conversion behavior that can cause unexpected and subtle errors. The most common error is that a system object containing a letter **I** or **i** is not found.

For more information about Turkish character sets and collations, see "Turkish character sets and collations" on page 439.

Character set conversion and client APIs

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

When working in a multi-character set environment, character set conversion issues can occur and it can be difficult to determine where the conversion issue occurred. When encountering character set conversion issues for client APIs, examine the database and connection options and properties that control character set conversion.

There are two categories into which conversion issues can be placed. The first involves sending data in the wrong format to the client API. Although this cannot happen with Unicode APIs, it is possible with all other client APIs and results in garbage data.

The second category of issue involves a character that does not have an equivalent in the final character set, or in one of the intermediate character sets. In this case, a substitution character is used. This is called lossy conversion and can happen with any client API. You can avoid lossy conversions by configuring the database to use UTF-8 for the database character set. See "Lossy conversion and substitution characters" [*SQL Anywhere Server - SQL Reference*].

Option and property settings that impact character set conversion

Database and connection options and properties in effect for a connection is typically available in the full connection string. However, you can also query for the settings using system functions such as PROPERTY, DB_PROPERTY, and CONNECTION_PROPERTY. For example:

Query	Description
SELECT PROPERTY('CharSet');	Returns the database server's operating system character set.
SELECT DB_PROPERTY('CharSet');	Returns the database CHAR character set.
SELECT DB_PROPERTY('NcharCharSet');	Returns the database NCHAR character set.
SELECT DB_PROPERTY('MultibyteCharSet');	Returns whether CHAR data uses a multibyte character set (On=yes, Off=no).
SELECT CONNECTION_PROPERTY('CharSet');	Returns the client CHAR character set.
SELECT CONNECTION_PROPERTY('NcharCharSet');	Returns the client NCHAR character set.
SELECT CONNECTION_PROPERTY('on_charset_conversion_failure');	Returns the value of the on_charset_conversion_warning option.

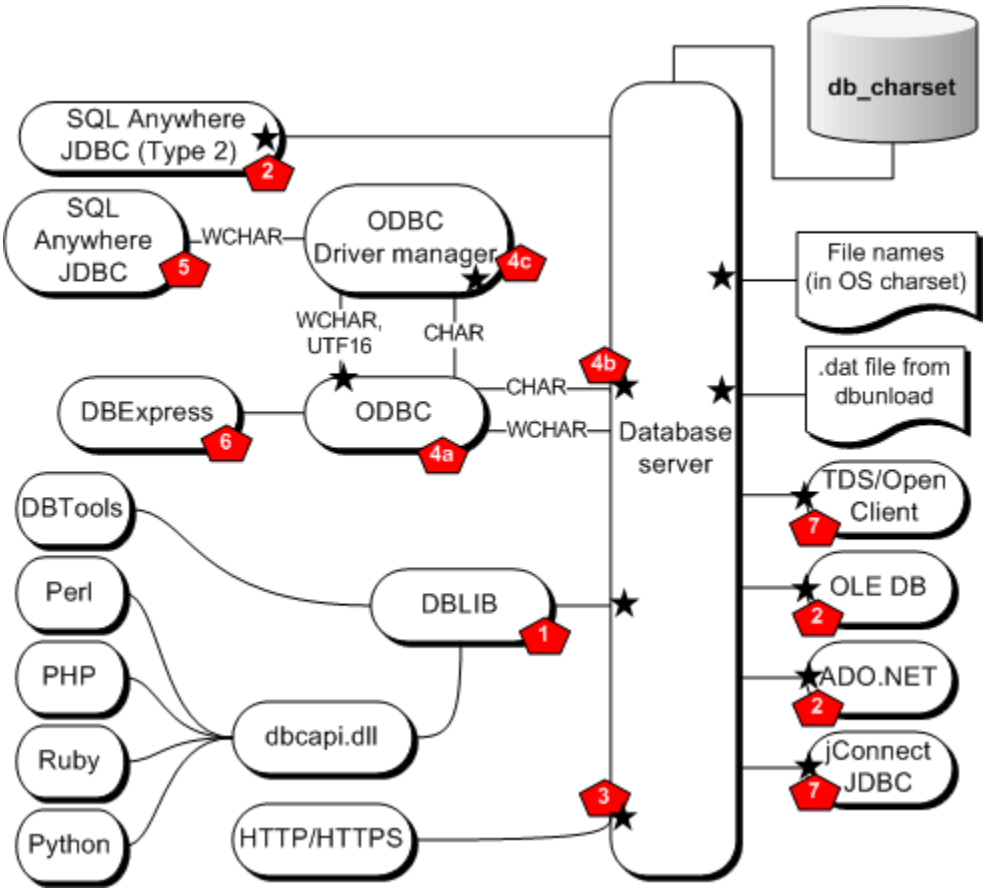
See also:

- "PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]
- "DB_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]
- "CONNECTION_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]
- "on_charset_conversion_failure option" on page 565

Client API and points of character set conversion

The following diagram shows the possible locations of character set conversion when client APIs interact with the database server.

The black stars in the diagram indicate a location where character set conversion can take place. The numbers in the diagram correspond to additional notes located after the diagram.



- **1 - DBLIB** DBLIB CHAR and NCHAR character sets default to the client operating system character set. The CHAR character set is used for all string data except BINARY and NCHAR. The NCHAR character set is used for NCHAR typed host variables and host parameters (these are not available from the DBTools or DBCAPI interfaces). The database server performs all character set conversion.

  You can set the CHAR character set and the NCHAR character set using the CharSet connection parameter and the db_change_nchar_charset function, respectively. See "CharSet (CS) connection parameter" on page 270 and "db_change_nchar_charset function" [*SQL Anywhere Server - Programming*].

- **2 - Unicode APIs** With Unicode APIs, a driver uses the Unicode UTF-16 encoding for character data, and converts Unicode data to/from the database character set. The driver converts Unicode host

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

parameters to the NCHAR character set (UTF-8) before sending them to the database server. Result set columns that are described as NCHAR are fetched in the NCHAR character set (UTF-8) and converted to Unicode after they are received. Setting the CharSet connection parameter is not recommended because it can result in a lossy conversion. See "Lossy conversion and substitution characters" [*SQL Anywhere Server - SQL Reference*].

- **3 - HTTP/HTTPS** **HTTP Services (SQL Anywhere as the server):** For HTTP services, there are two types of request: URL-encoded or multipart/form-data requests.

  URL-encoded: Requests take the form of application/x-www-form-urlencoded where variables are passed as key/value pairs. The database server automatically decodes %-encoded data (UTF-8 or database character set) and translates the key/value pairs into the database character set. Processed values can be extracted using the HTTP_VARIABLE function where the @BINARY or @TRANSPORT attributes can be used to return a value that is %-decoded but not character set translated, or the raw HTTP (transport) value, respectively.

  Multipart/form-data: Requests are considered to be binary—that is, using attributes @BINARY or @TRANSPORT would return identical values.

  Character set conversion of an HTTP service response is controlled by the CharsetConversion and AcceptCharset HTTP options, which you can set using the sa_set_http_option system procedure. See "sa_set_http_option system procedure" [*SQL Anywhere Server - SQL Reference*].

  Other settings which can impact character set conversion for requests and responses are the Accept-Charset header, which is used to specify the preferred character sets, and the Content-Type header, which is used to identify the character set of the content.

  HTTP stored procedures (SQL Anywhere as the client): For HTTP stored procedures, CHAR data is sent in the database character set. If any parameter is an NCHAR type, then all data is sent as UTF-8 (all CHAR parameters are translated to UTF-8). The request sends an Accept-Charset HTTP header identifying the database character set as the preferred character set. UTF-8 is always included in the list of preferred character sets. If the response specifies a character set in its Content-Type header that is not the database character set, then the client translates the response into the database character set.

- **4a - ODBC with Unicode entry points** If an ODBC application uses the Unicode entry points, it is considered a Unicode client API. WCHAR data is handled in the same way it is handled for Unicode APIs. If an ODBC application uses the Unicode entry points and CHAR (ANSI) data, the data is assumed to be in the database character set. Mixing use of CHAR and WCHAR data in one application is not recommended.

- **4b - ODBC with ANSI entry points** If an ODBC application uses the ANSI entry points, it is considered an ANSI client API. The CHAR character set defaults to the client OS character set. The CHAR character set can be changed using the CharSet connection parameter. See "CharSet (CS) connection parameter" on page 270.

  Any conversion of ANSI data is done by the database server. Fetching into WCHAR host variables can result in a lossy conversion for CHAR data on the database server. Mixing use of CHAR and WCHAR data in an application is not recommended.

- **4c - ODBC driver managers** Some ODBC driver managers convert all CHAR data to WCHAR data and then call the Unicode entry points.

- **5 - SQL Anywhere JDBC driver** The SQL Anywhere JDBC driver does not perform character set conversion and uses the Unicode entry points and WCHAR types of the ODBC driver. This is the same case as described for ODBC with Unicode entry points.

- **6 - Borland DBExpress** The Borland DBExpress driver uses ODBC as an ANSI client API. The CHAR character set defaults to the client OS character set. While you can change this using the CharSet connection parameter, it is not recommended because it can result in incorrect data.

- **7 - TDS clients** TDS clients such as Sybase Open Client and jConnect negotiate with the database server at connect time and prefer that character set conversion take place on the client. During the negotiation, the database server is instructed not to do character set conversion.

See also

- "Data type conversions" [*SQL Anywhere Server - SQL Reference*]
- "Lossy conversion and substitution characters" [*SQL Anywhere Server - SQL Reference*]
- "Converting NCHAR to CHAR" [*SQL Anywhere Server - SQL Reference*]
- "SQL Anywhere C API support" [*SQL Anywhere Server - Programming*]
- "Perl DBI support" [*SQL Anywhere Server - Programming*]
- "Python support" [*SQL Anywhere Server - Programming*]
- "SQL Anywhere PHP extension" [*SQL Anywhere Server - Programming*]
- "SQL Anywhere Ruby API support" [*SQL Anywhere Server - Programming*]
- "Sybase Open Client support" [*SQL Anywhere Server - Programming*]
- "Using SQL Anywhere as an HTTP web server" [*SQL Anywhere Server - Programming*]
- "Introduction to OLE DB" [*SQL Anywhere Server - Programming*]
- "SQL Anywhere .NET Data Provider" [*SQL Anywhere Server - Programming*]
- "ODBC support" [*SQL Anywhere Server - Programming*]
- "JDBC support" [*SQL Anywhere Server - Programming*]

Understanding locales

Both the database server and the client library recognize their language and character set environment using a **locale definition**.

Introduction to locales

The application locale, or client locale, is used by the client or client library when making requests to the database server, to determine the character set in which results should be returned, and the language of error messages, warnings, and other messages. The database server compares its own locale with the application locale to determine whether character set conversion is needed. Different databases on a server may have different locale definitions, and each client may have its own locale.

The locale consists of the following components:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Language** The language is a two-character string using the ISO 639-1 standard values (for example, DE for German). Both the database server and the client have language values for their locale.

  The database server uses the locale language to determine the language libraries to load. When creating a database, if no collation is specified, the database server also uses the language, together with the character set, to determine which collation to use.

  The client library uses the locale language to determine the language libraries to load, and the language to request from the database. See "Understanding the locale language" on page 417.

- **Character set** The character set is the code page, or encoding, in use. The client and server both have character set values, and they may differ. If they differ, character set conversion is used to enable interoperability. See "Understanding the locale character set" on page 418.

Understanding the locale language

The locale language is the language being used by the user of the client application, or expected to be used by users of the database server. For more information about how to find locale settings, see "Determining locale information" on page 431.

The client library, and the database server, both determine the language component of the locale in the same manner:

1. Use the value of the SALANG environment variable, if it exists. See "SALANG environment variable" on page 384.

2. For Windows, if the SALANG environment variable doesn't exist, check the SQL Anywhere language registry entry. See "Registry settings on installation" on page 398.

3. Check the operating system language setting.

4. If the language still cannot be determined by the above settings, default to English.

Language label values

The following table displays the valid language label values, together with the equivalent ISO 639 language codes:

Language	ISO 639-1 language code	Language label	Alternative label
Arabic	AR	arabic	N/A
Czech	CS	czech	N/A
Danish	DA	danish	N/A
Dutch	NL	dutch	N/A

Language	ISO 639-1 language code	Language label	Alternative label
English	EN	us_english	english
Finnish	FI	finnish	N/A
French	FR	french	N/A
German	DE	german	N/A
Greek	EL	greek	N/A
Hebrew	HE	hebrew	N/A
Hungarian	HU	hungarian	N/A
Italian	IT	italian	N/A
Japanese	JA	japanese	N/A
Korean	KO	korean	N/A
Lithuanian	LT	lithuanian	N/A
Norwegian	NO	norwegian	norweg
Polish	PL	polish	N/A
Portuguese	PT	portuguese	portugue
Russian	RU	russian	N/A
Simplified Chinese	ZH	chinese	simpchin
Spanish	ES	spanish	N/A
Swedish	SV	swedish	N/A
Thai	TH	thai	N/A
Traditional Chinese	TW	tchinese	tradchin
Turkish	TR	turkish	N/A
Ukrainian	UK	ukrainian	N/A

Understanding the locale character set

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Both application and server locale definitions have a character set. The application uses its character set when requesting character strings from the database server. The database server compares the database character set with that of the application to determine whether character set conversion is needed. If the database server cannot convert to and from the client character set, the connection fails.

1. If the SACHARSET environment variable is set, its value is used to determine the character set. See "SACHARSET environment variable" on page 382.

   The database server uses SACHARSET only when creating new databases, and then only if no collation is specified.

2. If the connection string specifies a character set, it is used. For more information, see "CharSet (CS) connection parameter" on page 270.

3. Sybase Open Client applications check the *locales.dat* file in the *locales* subdirectory of the *Sybase* release directory.

4. Character set information from the operating system is used to determine the locale:

   - On Windows operating systems, the current Windows ANSI code page is used.

   - On Unix platforms, the following Locale Environment Variables are examined, in the specified order: LC_ALL, LC_MESSAGES, LC_CTYPE, LANG. For the first of these environment variables found to be set, its value is used to determine the character set. If the character set cannot be determined from the operating system, the default of iso_1 (also referred to as Windows code page 28591, ISO 8859-1 Latin I, ISO 8859-1 Latin-1, or iso_8859-1:1987) is used.

5. On any other platform, a default code page cp1252 is used.

For more information about how to find locale settings, see "Determining locale information" on page 431.

Understanding collations

A collation describes how to sort and compare characters from a particular character set or encoding. SQL Anywhere supports two collation algorithms: the SQL Anywhere Collation Algorithm (SACA), and the Unicode Collation Algorithm (UCA). SACA provides fast, compact, and reasonable sorting at the expense of linguistic correctness. UCA provides linguistic correctness, but with a small expense in storage requirements and execution time.

This section describes the supplied collations, and provides suggestions about which collations to use.

For more information about how to create a database with a specific collation, see "Create a database with a named collation" on page 433 and "Initialization utility (dbinit)" on page 799.

For information about customization of the UCA collation using collation tailoring syntax, see "Collation tailoring options" on page 427.

SQL Anywhere Collation Algorithm (SACA)

The SQL Anywhere Collation Algorithm (SACA) provides reasonable comparison, ordering, and case conversion of single-byte and multibyte character sets. The algorithm is space efficient and fast. The mapped form of a string, such as an index, is the same length as the original string. The mappings for comparison, ordering, and case conversion use a simple table lookup of each byte value of the string.

The SACA has been provided with SQL Anywhere since its early days as Watcom SQL.

Single-byte character sets

In a typical collation for a single-byte character set, all accented and unaccented forms of a character are mapped to the same value, making the collation accent insensitive. Accented and unaccented forms of the same letter compare as exactly equal and sort near each other.

The collation also provides conversion between uppercase and lowercase letters, preserving accents.

Multibyte character sets

In multibyte character sets, the lead-bytes are mapped into the 256 distinct values. Follow bytes are compared using their binary value.

For most collations for multibyte character sets, this mapping technique provides a reasonable ordering because the character set encoding groups characters into 256-byte pages identified by the lead byte. The pages, and the characters within each page, are in a reasonable order in the character set. The collations typically preserve the ordering of the pages (lead bytes) within the character set. Some pages may be ordered by other characteristics. For example, the 932JPN collation provided for Japanese code page 932 groups the full-width (Kanji) and half-width (katakana) characters.

Case conversion is provided only for the 7-bit English characters.

UTF-8 character sets

UTF-8 is a multibyte character set. Each character contains from one to four bytes. SQL Anywhere provides the UTF8BIN collation for sorting UTF-8 characters.

In UTF8BIN, lead bytes are mapped into 256 distinct values, and follow bytes are compared using their binary values. Because of the representation of characters in UTF-8 and the limitation of 256 distinct mapping values, it is not possible to group related characters such as accented and unaccented forms of the same letter. The ordering is essentially binary.

Case conversion is supported only for the 7-bit English characters.

Unicode Collation Algorithm (UCA)

The Unicode Collation Algorithm (UCA) is an algorithm for sorting the entire Unicode character set. It provides linguistically correct comparison, ordering, and case conversion. The UCA was developed as part of the Unicode standard. SQL Anywhere implements the UCA using the International Components for Unicode (ICU) open source library, developed and maintained by IBM.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Note
The default UCA ordering sorts most characters in most languages into an appropriate order. However, because of the sorting and comparison variations between languages sharing characters, the UCA cannot provide proper sorting for all languages. For this purpose, ICU provides a syntax for tailoring the UCA. See "Collation tailoring options" on page 427.

The UCA provides advanced comparison, ordering, and case conversion at a small cost in space and time.

The mapped form of a string is longer than the original string. The algorithm provides sophisticated handling of more complex characters.

Unlike the SQL Anywhere Collation Algorithm (SACA) the Unicode Collation Algorithm (UCA) is only for use with single-byte and UTF-8 character sets, and it separates each character into one or more attributes. For letters, these attributes are base character, accent, and case.

Non-letters typically have only one attribute, the base character.

UCA compares character strings as follows:

- Compare the base characters. If one string of base characters differs from the other, then the comparison is complete. Accent and case are not considered.
- If the database is accent sensitive, compare the accents. If the accents differ, then the comparison is complete. Case is not considered.
- If the database is case sensitive, compare the case of each character.

The original string values are equal if and only if the base characters, accents, and case are the same for both strings.

Example

Suppose UCA is used to compare the strings in the first column of the table below. The subsequent columns describe the three attributes for each string. Notice that the base characters are identical; the words differ only in accents and case.

String	Base characters	Accents	Case
noel	noel	none, none, none, none	lower, lower, lower, lower
noël	noel	none, none, accent, none	lower, lower, lower, lower
Noel	noel	none, none, none, none	upper, lower, lower, lower
Noël	noel	none, none, accent, none	upper, lower, lower, lower

The following table shows the ordering that would occur in the four possible combinations of accent- and case-sensitivity using UCA:

Accent sensitive	Case sensitive	ORDER BY result	Explanation
N	N	Noel, noël, Noël, noel in any order	• Accents ignored • Case ignored • All values considered equal • Random order within set of four
Y	N	Noel, noel in any order, followed by noël, Noël in any order	• No-accents before accents, so e before ë • Case ignored, N and n are in random order within each set of two
N	Y	Noel, Noël in any order, followed by noël, noel in any order	• Uppercase before lowercase, so N before n • Accents ignored, e and ë are in random order within each set of two
Y	Y	Noel noel Noël noël	• No-accents before accents, so e before ë • Uppercase before lowercase, so N before n

Collations in a SQL Anywhere database

CHAR collation

CHAR data types, including CHAR, VARCHAR, and LONG VARCHAR, can use a collation that uses the SQL Anywhere Collation Algorithm (SACA) or they can use the Unicode Collation Algorithm (UCA). In either case, the collation used is referred to as the CHAR collation.

NCHAR collation

NCHAR data types, including NCHAR, NVARCHAR, and LONG NVARCHAR, can use the Unicode Collation Algorithm (UCA) or can use the UTF8BIN collation, which uses the SQL Anywhere Collation Algorithm (SACA).

Choosing case and accent sensitivity

When a SQL Anywhere database is created, if case sensitivity is not specified, then it is case insensitive. It can be made case sensitive by specifying the appropriate option. It is not possible to change the case sensitivity after the database has been created without rebuilding the database.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The case sensitivity for the database determines the case sensitivity for both the SACA and UCA collations, and so it also determines the case sensitivity of both the CHAR and NCHAR collations.

When a SQL Anywhere database is created, if accent sensitivity is not specified, then it is accent insensitive. It can be made accent sensitive by specifying the appropriate option. It is not possible to change the accent sensitivity after the database has been created without rebuilding the database.

The accent sensitivity for the database affects only the UCA collation, whether it is used for the CHAR or NCHAR collations or both. If you choose SACA collations for both CHAR and NCHAR collations, then the options for accent sensitivity have no effect. Accent sensitivity is an attribute of SACA collations and cannot be specified using the options provided when creating the database.

Supported and alternate collations

The following table lists compatibility collations introduced in SQL Anywhere version 8 that can be used with the SORTKEY and COMPARE functions.

In SQL Anywhere version 10, implementation of the SORTKEY and COMPARE functions changed to ICU (International Components for Unicode) and Unicode Collation Algorithm (UCA) and collation names now map to the UCA collation. Sorting and comparison in SQL Anywhere version 10 and later may not be identical to earlier versions of SQL Anywhere.

Collation label	Description
874THAIBIN	Code Page 874, Windows Thai, ISO8859-11, binary ordering
932JPN	Code Page 932, Japanese Shift-JIS with Microsoft extensions
936ZHO	Code Page 936, Simplified Chinese, PRC GBK 2312-80 8-bit encoding
949KOR	Code Page 949, Korean KS C 5601-1987 encoding, Wansung
950ZHO_HK	Code Page 950, Traditional Chinese, Big 5 encoding with HKSCS
950ZHO_TW	Code Page 950, Traditional Chinese, Big 5 encoding
1250LATIN2	Code Page 1250, Windows Latin 2, Central/Eastern European
1250POL	Code Page 1250, Windows Latin 2, Polish
1251CYR	Code Page 1251, Cyrillic
1252LATIN1	Code Page 1252, Windows Latin 1, Western
1252NOR	Code Page 1252, Windows Latin 1, Norwegian
1252SPA	Code Page 1252, Windows Latin 1, Spanish

Collation label	Description
1252SWEFIN	Code Page 1252, Windows Latin 1, Swedish/Finnish
1253ELL	Code Page 1253, Windows Greek, ISO8859-7 with extensions
1254TRK	Code Page 1254, Windows Latin 5, Turkish, ISO 8859-9 with extensions
1254TRKALT	Code Page 1254, Windows Turkish, ISO8859-9 with extensions, I-dot equals I-no-dot
1255HEB	Code Page 1255, Windows Hebrew, ISO8859-8 with extensions
1256ARA	Code Page 1256, Windows Arabic, ISO8859-6 with extensions
1257LIT	Code Page 1257, Lithuanian
EUC_CHINA	Simplified Chinese GB 2312-80 Encoding
EUC_JAPAN	Japanese EUC JIS X 0208-1990 and JIS X 0212-1990 Encoding
EUC_KOREA	Korean KS C 5601-1992 Encoding, Johab
EUC_TAIWAN	Taiwanese Big 5 Encoding
ISO1LATIN1	ISO8859-1, ISO Latin 1, Western, Latin 1 Ordering
ISO9LATIN1	ISO8859-15, ISO Latin 9, Western, Latin 1 Ordering
ISO_1	ISO8859-1, Latin 1, Western
ISO_BINENG	Binary ordering, English ISO/ASCII 7-bit letter case mappings
UCA	Standard default UCA collation
UTF8BIN	UTF-8, 8-bit multibyte encoding for Unicode, binary ordering

Alternate collations

Alternate collations are available for compatibility with older versions of SQL Anywhere, or for special purposes. To see the full list of supported alternate collations, run the following command:

```
dbinit -l+
```

Adaptive Server Enterprise collations

The following table lists the supported Adaptive Server Enterprise collations for use with features such as the SORTKEY function.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Description	Collation name	Collation ID
Default Unicode multilingual	default	0
CP 850 Alternative: no accent	altnoacc	39
CP 850 Alternative: lowercase first	altdict	45
CP 850 Western European: no case, preference	altnocsp	46
CP 850 Scandinavian dictionary	scandict	47
CP 850 Scandinavian: no case, preference	scannocp	48
GB Pinyin	gbpinyin	n/a
Binary sort	binary	50
Latin-1 English, French, German dictionary	dict	51
Latin-1 English, French, German no case	nocase	52
Latin-1 English, French, German no case, preference	nocasep	53
Latin-1 English, French, German no accent	noaccent	54
Latin-1 Spanish dictionary	espdict	55
Latin-1 Spanish no case	espnocs	56
Latin-1 Spanish no accent	espnoac	57
ISO 8859-5 Russian dictionary	rusdict	58
ISO 8859-5 Russian no case	rusnocs	59
ISO 8859-5 Cyrillic dictionary	cyrdict	63
ISO 8859-5 Cyrillic no case	cyrnocs	64
ISO 8859-7 Greek dictionary	elldict	65
ISO 8859-2 Hungarian dictionary	hundict	69
ISO 8859-2 Hungarian no accents	hunnoac	70
ISO 8859-2 Hungarian no case	hunnocs	71
ISO 8859-5 Turkish dictionary	turdict	72

Description	Collation name	Collation ID
ISO 8859-5 Turkish no accents	turnoac	73
ISO 8859-5 Turkish no case	turnocs	74
CP 874 (TIS 620) Royal Thai dictionary	thaidict	1
ISO 14651 ordering standard	14651	22
Shift-JIS binary order	sjisbin	179
Unicode UTF-8 binary sort	utf8bin	24
EUC JIS binary order	eucjisbn	192
GB2312 binary order	gb2312bn	137
CP932 MS binary order	cp932bin	129
Big5 binary order	big5bin	194
EUC KSC binary order	euckscbn	161

See also

- "COMPARE function [String]" [*SQL Anywhere Server - SQL Reference*]
- "SORTKEY function [String]" [*SQL Anywhere Server - SQL Reference*]
- "What is ICU, and when is it needed?" on page 403

Choosing collations

When you create a database, SQL Anywhere can choose a default collation based on operating system language and character set settings. Usually the default collation is a suitable choice, but you can also explicitly choose a collation to match your needs from the wide selection of supplied collations. SQL Anywhere can support more than one collation for a particular language.

You should choose a collation that uses a character set and sort order that are appropriate for the data in your database. You can also specify collation tailoring options for additional control over the sorting and comparing of characters. For more information about creating databases, see "Creating a SQL Anywhere database" on page 5.

For more information about data sorting and international features, see "International languages and character sets" on page 400.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- “Create a database with a named collation” on page 433
- “Recommended character sets and collations” on page 436
- “CREATE DATABASE statement” [*SQL Anywhere Server - SQL Reference*]
- “Initialization utility (dbinit)” on page 799

Considerations when choosing a collation

When choosing the collation for your database, consider the following:

- There is a performance cost, and extra complexity in system configuration, when you use character set conversion. Choose a collation that avoids the need for character set conversion. Character set conversion is not used if the database server and client use the same character set.

  You can avoid character set conversion by using a collation sequence in the database that matches the character set in use on your client computer operating system. Choose the ANSI character set for Windows operating systems on the client computer.

- If your client computers use a variety of characters sets, or if the database must store Unicode data, consider using the UCA and/or UTF8BIN collations. However, note that the UCA collation cannot be used with multibyte character sets other than UTF-8.

- Choose a collation that uses a character set and sort order appropriate for the data in the database. It is often the case that there are several collations that meet this requirement.

Collation tailoring options

If you choose the UCA collation when you create a database, you can optionally specify collation tailoring options. If you do not choose UCA as the collation, you can still use tailoring syntax to control case sensitivity. You can also specify tailoring options when comparing or sorting data using the COMPARE and SORTKEY functions.

Swedish collation tailoring

To tailor a UCA collation to conform to the Swedish Academy's 2005 standards in which V and W are considered to be different characters at the primary level, specify `UCA (locale=swe;sorttype=phonebook)`. Without sorttype=phonebook, V and W are considered to be the same character in the Swedish locale.

Japanese collation tailoring

UCA distinguishes some Hiragana and Katakana letters only at the tertiary level and those differences are lost in case insensitive collations. To tailor a UCA collation to define primary level differences between all Hiragana letters, as well as primary-level differences between all Katakana letters, specify `UCA (locale=ja;sorttype=direct;...)`. Although these tailoring options do not provide absolute correct sorting semantics, they do provide correct equality semantics.

Summary of collation tailoring options

Collation tailoring options take the form of keyword-value pairs. Following is a table of the supported keywords, including their allowed alternate forms, and their allowed values.

Note
Databases created with collation tailoring options cannot be started using a pre-10.0.1 database server.

Keyword	Collation	Alternate forms	Allowed values
Locale	UCA	(none)	Any valid locale code. For example, en.
CaseSensitivity	All supported collations	CaseSensitive, Case	• **respect** Respect case differences between letters. For the UCA collation, this is equivalent to UpperFirst. For other collations, it depends on the collation itself. • **ignore** Ignore case differences between letters. • **UpperFirst** Always sort uppercase first (Aa). • **LowerFirst** Always sort lowercase first (aA).
AccentSensitivity	UCA	AccentSensitive, Accent	• **respect** Respect accent differences between letters. • **ignore** Ignore accent differences between letters. • **French** Respect accent sensitivity with French rules.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Keyword	Collation	Alternate forms	Allowed values
PunctuationSensitivity	UCA	PunctuationSensitive, Punct	• **ignore** Ignore differences in punctuation. • **primary** Use first level sorting (consider letter, only). For example, a > b. • **quaternary** Use fourth level sorting: consider letter first, then case, then accent, and then punctuation. For example, multiByte, multibyte, multi-byte, and multi-Byte, are sorted as: ○ multiByte ○ multibyte ○ multi-Byte ○ multi-byte You cannot specify quaternary with a case- or accent-insensitive database.

Keyword	Collation	Alternate forms	Allowed values
SortType	UCA	(none)	The type of sort to use. Possible values include: • phonebook • traditional • standard • pinyin • stroke • direct • posix • big5han • gb2312han For more information about these sort types, see Unicode Technical Standard #35 at http://www.unicode.org/reports/tr35/.

See also

- "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "Initialization utility (dbinit)" on page 799
- "COMPARE function [String]" [*SQL Anywhere Server - SQL Reference*]
- "SORTKEY function [String]" [*SQL Anywhere Server - SQL Reference*]

How SQL Anywhere chooses the default collation for a new database

When a new database is created, and the collation is not explicitly specified, SQL Anywhere uses the language and character set to determine the collation.

- The language comes from the SALANG environment variable (if it exists), the registry, or the operating system. See "SALANG environment variable" on page 384.
- The character set comes from the SACHARSET environment variable (if it exists) or the operating system. See "SACHARSET environment variable" on page 382.

International language and character set tasks

This section groups together the tasks associated with international language and character set issues.

Determining the default collation

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

If you do not explicitly specify a collation when creating a database, a default collation is used. The default collation depends on the operating system you are working on.

To determine the default collation for your computer

1. Start Interactive SQL.
2. Connect to the sample database.
3. Enter the following query:

   ```
   SELECT PROPERTY( 'DefaultCollation' );
   ```

   The default collation is returned.

   For more information about this collation, see "Choosing collations" on page 426.

Determining locale information

You can determine locale information using functions such as PROPERTY, DB_PROPERTY, and CONNECTION_PROPERTY. The following table shows how to use these functions to return locale information about the client connection, database, and database server.

System function and parameter	Return value
`SELECT PROPERTY( 'CharSet' );`	Character set of the database server. Usually the character set of the computer hosting the database server.
`SELECT PROPERTY( 'DefaultCollation' );`	Default CHAR collation used by the database server for creating databases.
`SELECT PROPERTY( 'DefaultNcharCollation' );`	Default NCHAR collation used by the database server for creating databases.
`SELECT PROPERTY( 'Language' );`	The locale language for the database server.
`SELECT DB_PROPERTY( 'CharSet' );`	Character set used to store CHAR data in the database.
`SELECT DB_PROPERTY( 'NcharCharSet' );`	Character set used to store NCHAR data in the database.
`SELECT DB_PROPERTY( 'MultiByteCharSet' );`	Whether CHAR data uses a multibyte character set (On=yes, Off=no).

System function and parameter	Return value
`SELECT DB_PROPERTY( 'Language' );`	Comma-separated list of two-letter codes representing the languages supported by database CHAR collation.
`SELECT DB_PROPERTY( 'Collation' );`	CHAR collation name in use by the database server.
`SELECT DB_PROPERTY( 'NcharCollation' );`	NCHAR collation name in use by the database server.
`SELECT CONNECTION_PROPERTY( 'CharSet' );`	Client's CHAR data character set.
`SELECT CONNECTION_PROPERTY( 'NcharCharSet' );`	Character set of NCHAR data for the connection.
`SELECT CONNECTION_PROPERTY( 'Language' );`	Client language for the connection.

See also

- "PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]
- "DB_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]
- "CONNECTION_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]

Setting locales

You can use the default locale on your operating system, or explicitly set a locale for use by the SQL Anywhere components on your computer.

To set the SQL Anywhere locale

1. If the default locale is appropriate for your needs, you do not need to take any action.

   For more information about how to find out the default locale of your operating system, see "Determining locale information" on page 431.

2. If you need to change the locale, you can set either or both of the SALANG and SACHARSET environment variables:

   ```
   SACHARSET=charset SALANG=language-code
   ```

   The *charset* is a valid character set label, and *language-code* is a language code from the list of valid languages. See "Language label values" on page 417.

   For more information about setting environment variables on different operating systems, see "SQL Anywhere environment variables" on page 377.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Create a database with a named collation

You can specify the collation for each database when you create the database. The default collation is inferred from the code page and language of the database server's computer's operating system.

For information about using the NCHAR collation, see "NCHAR collation" on page 422.

To specify a database collation when creating a database (command line)

1. Run the following command to list the recommended collation sequences:

   ```
   dbinit -l
   ```

   The first column of the list is the collation label, which you supply when creating the database.

2. Create a database using the dbinit utility, specifying a collation sequence using the -z option. The following command creates a database with a Greek collation.

   ```
   dbinit -z 1253ELL mydb.db
   ```

   The following command creates a case sensitive database, *spanish.db*, which uses the 1262spa collation for non-NCHAR data. For NCHAR data, the UCA collation is specified, with locale es, and sorting by lowercase first.

   ```
   dbinit -c -z 1252spa -zn uca(locale=es;case=LowerFirst) spanish.db
   ```

To specify a database collation when creating a database (SQL)

- You can use the CREATE DATABASE statement to create a database. The following statement creates a database with a Greek collation:

  ```
  CREATE DATABASE 'mydb.db' COLLATION '1253ELL';
  ```

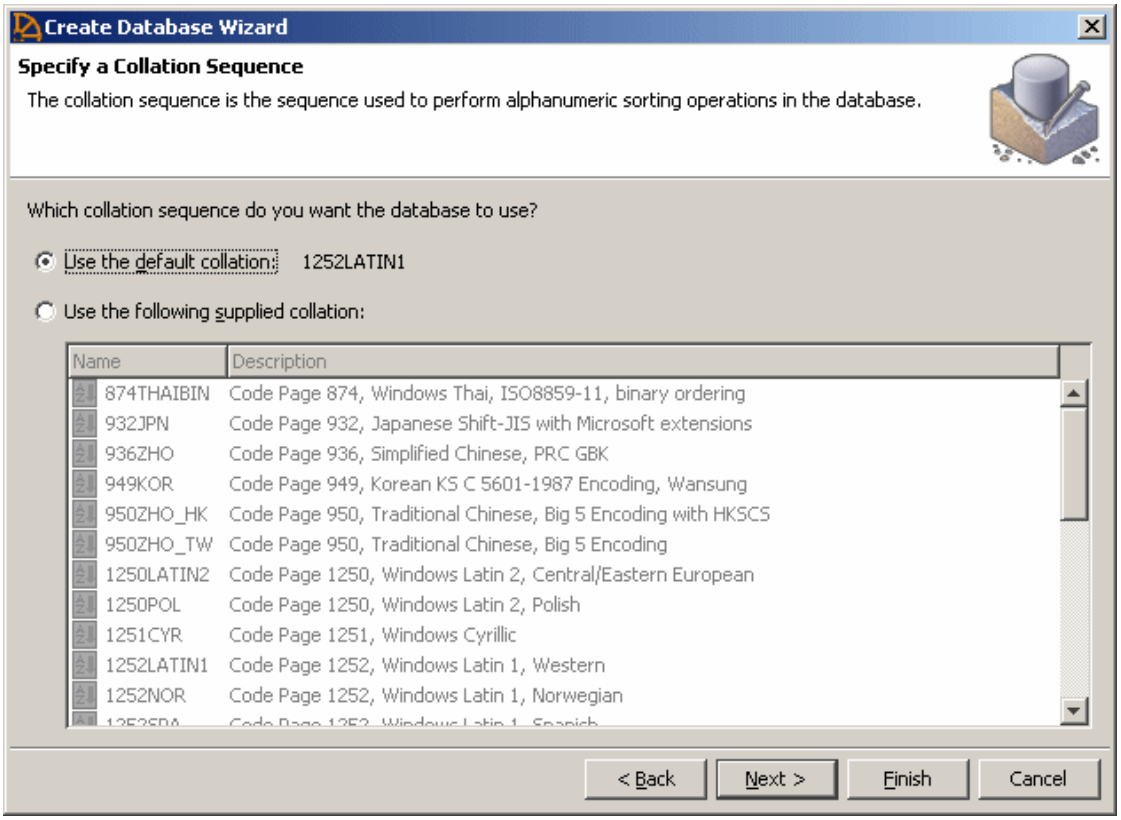
  The following statement creates a database using code page 1252 and uses the UCA for both CHAR and NCHAR data types. Accents and case are respected during comparison and sorting.

  ```
  CREATE DATABASE 'c:\\uca.db' COLLATION 'UCA' ENCODING 'CP1252' NCHAR
  COLLATION 'UCA' ACCENT RESPECT CASE RESPECT;
  ```

To specify a database collation when creating a database (Sybase Central)

- Choose **Tools** » **SQL Anywhere 12** » **Create Database**.

  The **Create Database Wizard** has a page where you choose a collation from a list.



See also

- "Tutorial: Creating a SQL Anywhere database" on page 11
- "Creating a SQL Anywhere database" on page 5
- "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "Initialization utility (dbinit)" on page 799

Change a database from one collation to another

Changing a database to another collation requires a rebuild of the database. Collations are chosen at database creation time and cannot be changed.

To change collations

1. Determine the character set for the existing database as follows:

   ```
   SELECT DB_PROPERTY( 'CharSet' );
   ```

   For early versions of SQL Anywhere, this property may not exist. The character set is also implied by the collation name. For example, collation 1252LATIN1 uses code page 1252.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

2. Determine the character set for the data in the existing database.

   This should be the same as, or compatible with, the database character set. If it is not, it is an excellent reason to rebuild the database, but requires great care in the rebuilding process.

   In particular, if you have been using a database with collation 850LATIN1 with earlier versions of SQL Anywhere that either did not support character set conversion (versions 5 and earlier) or disabled it by default (versions 6 and 7), and if your client applications were normal Windows applications, you may have code page 1252 character data in your database that is expecting data to be in code page 850. A simple test for this case is to use UNLOAD TABLE with the ENCODING option to unload some character data, then view it in Windows Notepad. If accented data is correct, then the character data in the database matches the Windows ANSI code page, which for English and other Western European languages is code page 1252. If the data appears correct in a DOS-based editor, then the character data matches the Windows OEM code page, which is likely 437 or 850.

3. Unload the database.

   If the data character set is incompatible with the database character set, it is critical that the data be unloaded without character set conversion. Depending on the version of SQL Anywhere being used, you can use the internal unload feature of dbunload, or manually unload the data using the UNLOAD TABLE statement.

4. Create the new database, specifying the collations and character sets you want to use.

5. Load the data into the new database.

   If the unloaded data and schema (*reload.sql*) match the character set of the computer used to do the reload, you can use the external reload option of dbunload. The data is automatically converted to the correct character set for the database.

   If the data's encoding does not match the character set of the database, and you are loading data using LOAD TABLE statements (internal reload), you must use the ENCODING clause; the database server does not, by default, perform character set conversion for data loaded using LOAD TABLE statements.

   If the data's encoding does not match the code page of the computer on which you are working, and you are loading using INPUT statements (external reload), you must use the ENCODING clause; otherwise, the database server assumes that the data is in the computer's native character set.

See also

- "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "UNLOAD statement" [*SQL Anywhere Server - SQL Reference*]
- "INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "Create a database with a named collation" on page 433
- "Unload utility (dbunload)" on page 864
- "Rebuilding databases" [*SQL Anywhere Server - SQL Usage*]
- "CharSet (CS) connection parameter" on page 270

Character set and collation reference information

The following sections provide information about character sets and collations for SQL Anywhere.

Recommended character sets and collations

While SQL Anywhere recognizes the names of hundreds of character sets, code pages, encodings and collations, this section lists those that are recommended for use with Windows and Unix platforms, depending on the language in use.

Use the dbinit -le option to obtain a list of all the available character set labels for a SQL Anywhere database. Use the dbinit -l option to obtain a list of available collations for a SQL Anywhere database. See "Initialization utility (dbinit)" on page 799.

Where a character set encoding or label must be specified, use the value from the Character set label column or one of the labels listed by dbinit -le. Where a collation must be specified, use the value from the Collation or Alternate collation column or one of the labels listed by dbinit -l.

Note
On Mac OS X, and for languages not found in the tables below, the UTF-8 encoding should be used with collations UCA or UTF8BIN.

Windows platforms

Language	Windows Code Page	Character set label	Collation	Alternate collation
Arabic	1256	Windows-1256	1256ARA	
Central and Eastern European	1250	Windows-1250	1250LATIN2	
Danish	1252	Windows-1252	1252LATIN1	
Dutch	1252	Windows-1252	1252LATIN1	
English	1252	Windows-1252	1252LATIN1	
Finnish	1252	Windows-1252	1252SWEFIN	
French	1252	Windows-1252	1252LATIN1	
German	1252	Windows-1252	1252LATIN1	
Greek	1253	Windows-1253	1253ELL	
Hebrew	1253	Windows-1253	1255HEB	
Italian	1252	Windows-1252	1252LATIN1	

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Language	Windows Code Page	Character set label	Collation	Alternate collation
Japanese	932	Windows-31J	932JPN	
Korean	949	IBM949	949KOR	
Lithuanian	1257	Windows-1257	1257LIT	
Norwegian	1252	Windows-1252	1252NOR	
Polish	1250	Windows-1250	1250POL	
Portuguese	1252	Windows-1252	1252LATIN1	
Russian	1251	Windows-1251	1251CYR	
Simplified Chinese	936	GBK	936ZHO	
Spanish	1252	Windows-1252	1252SPA	
Swedish	1252	Windows-1252	1252SWEFIN	
Thai	874	TIS-620	874THAIBIN	
Traditional Chinese - Hong Kong	950	Big5-HKSCS	950ZHO_HK	
Traditional Chinese - Taiwan	950	Big5	950ZHO_TW	
Turkish	1254	Windows-1254	1254TRK	1254TRKALT
Ukrainian	1251	Windows-1251	1251CYR	
Western European	1252	Windows-1252	1252LATIN1	

Unix platforms

Language	Character set label	Collation	Alternate collation
Arabic	ISO_8859-6:1987	UCA	
Central and Eastern European	ISO_8859-2:1987	UCA	
Danish	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Dutch	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
English	ISO-8859-15	ISO9LATIN1	ISO1LATIN1

Language	Character set label	Collation	Alternate collation
Finnish	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
French	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
German	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Greek	ISO_8859-7:1987	UCA	
Hebrew	ISO_8859-8:1988	UCA	
Italian	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Japanese	EUC-JP[1]	EUC_JAPAN	
Korean	EUC-KR	EUC_KOREA	
Lithuanian	(use UTF-8)	UCA or UTF8BIN	
Norwegian	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Polish	ISO_8859-2:1987	UCA	
Portuguese	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Russian	ISO_8859-5:1988	UCA	
Simplified Chinese	GB2312	EUC_CHINA	
Spanish	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Swedish	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Thai	(use UTF-8)	UCA or UTF8BIN	
Traditional Chinese - Hong Kong	Big5-HKSCS	950ZHO_HK	950TWN
Traditional Chinese - Taiwan	EUC-TW	EUC_TAIWAN	
Traditional Chinese - Taiwan	Big5	950ZHO_TW	
Turkish	ISO_8859-9:1989	920TRK	
Ukrainian	ISO_8859-5:1988	UCA	
Western European	ISO-8859-15	ISO9LATIN1	ISO1LATIN1

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

[1] EUC-JP is an alternate label for the SQL Anywhere label Extended_UNIX_Code_Packed_Format_for_Japanese.

Turkish character sets and collations

The Turkish language has two forms of what appears to be the letter **I**. One form, referred to as **I-dot**, appears as the following:

```
i, İ
```

The second form, referred to as **I-no-dot**, appears as the following:

```
ı, I
```

Even though these letters appear as variations of the same letter, in the Turkish alphabet they are considered to be distinct letters. SQL Anywhere provides the Turkish collation 1254TRK to support these variations.

Turkish rules for case conversion of these characters are incompatible with ANSI SQL standard rules for case conversion. For example, Turkish says that the lowercase equivalent of `I` is:

```
ı
```

However, the ANSI standard says that it is:

```
i
```

For this reason, correct case-insensitive matching is dependent on whether the text being matched is Turkish or English/ANSI. In many contexts, there is not enough information to make such a distinction, which leads to some non-standard behaviors in such databases.

For example, consider the following statements, executed against a database using the 1254TRK collation:

```
SELECT * FROM syshistory   //actual table name is SYSHISTORY
SELECT * FROM fig          //actual name is FİG
```

The first statement references a system object, and ANSI SQL conversion rules are required to match the name. The second statement references a user object, and Turkish conversion rules are required to match the name. However, the database server cannot tell which conversion rules to use until it knows what the object is, and it cannot know what the object is until it knows what conversion rules to use. The situation cannot be resolved properly for both system and user objects. In this example, since the database server is using the Turkish collation 1254TRK, the first statement fails because lowercase I is not considered equivalent to uppercase I. The second statement succeeds.

The incompatibility of Turkish and ANSI standards requires that system object references in Turkish databases specify the object name in the correct case, that is, the case used to create the object. The first statement above should be written as follows:

```
SELECT * FROM SYSHISTORY
```

In fact, only the letter I must be in the correct case.

As an alternative, it is acceptable, although unusual, to write the statement as follows:

```
SELECT * FROM syshıstory   //I-no-dot
```

Note that keywords such as INSERT are case-insensitive even in Turkish databases. SQL Anywhere knows that all keywords use only English letters, so it uses ANSI case conversion rules when matching keywords. SQL Anywhere also applies this knowledge for certain other identifiers, such as built-in functions. However, objects whose names are stored in the catalog must be specified using the correct case or letter, as described above. To determine the correct case, look in the system view that defines the system object. See "System views" [*SQL Anywhere Server - SQL Reference*].

Data in case-insensitive Turkish databases

Similar rules govern data in case-insensitive Turkish databases. For example if a data value is

```
FİG
```

then a lowercase reference to that data should be

```
fig
```

Then, the same I-dot character is used in both forms.

Alternative Turkish collation 1254TRKALT

For some application developers, the Turkish letter I problem can cause significant problems. The correct solution is to ensure that all object references are in the proper case or that the proper form of the letter I is used. However, it may be more expedient to violate the Turkish rules in favor of the ANSI rules.

SQL Anywhere provides the collation 1254TRKALT, which is identical to 1254TRK, except that it makes I-dot and I-no-dot equivalent characters.

It is important to understand the consequences of this change. In a 1254TRKALT database, the following strings are equal:

```
fig
fıg
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

This is not correct for a Turkish user, but may be acceptable for other users.

The second issue appears when using ORDER BY. Consider the following strings:

```
ia
ıa
ıs
is
```

In a 1254TRK database, an ORDER BY of the strings would produce the following:

```
ıa
ıs
ia
is
```

because I-no-dot is less than I-dot. In a 1254TRKALT database, the order would be

```
ia
ıa
ıs
is
```

because I-no-dot is equal to I-dot.

Managing user IDs, authorities, and permissions

Permissions allow you to restrict access to objects and commands based on a user's identity and group membership. Users with access permission, such as the creator of the object, can choose to grant permissions on the object to other users or groups.

Database permissions and authorities

Each user of a database has a name they enter when connecting to the database (user ID), and they belong to at least one group. Users and groups also have authorities and permissions attributed to them that allow them to perform their tasks while maintaining the security and privacy of information within the database.

A **permission** grants the ability to perform a create, modify, query, use, or delete database objects such as tables, views, users, and so on. An **authority** grants the ability to perform a task at the database level, such as backing up the database, or performing diagnostic tracing. SQL Anywhere allows you to grant permissions and authorities to user and groups.

While all permissions are inheritable (from the groups to which the user belongs), only some authorities are inheritable.

Inheriting authorities

The following table lists the authorities you can assign to users, and whether they are inherited through group membership:

Authority	Inherited through group membership	More information
BACKUP	No	See “BACKUP authority” on page 444.
DBA	No	See “DBA authority” on page 444.
PROFILE	Yes	See “PROFILE authority” on page 445.
READCLIENTFILE	Yes	See “READCLIENTFILE authority” on page 446.
READFILE	Yes	See “READFILE authority” on page 446.
REMOTE DBA		See “REMOTE DBA authority” on page 446.
RESOURCE	No	See “RESOURCE authority” on page 447.
VALIDATE	No	See “VALIDATE authority” on page 447.
WRITECLIENTFILE	Yes	See “WRITECLIENTFILE authority” on page 447.

Inheriting permissions

The following table lists the permissions you can assign to users, and whether they are inherited through group membership:

Permission	Inherited through group membership		More information
ALL	Yes	Allows the user to perform all tasks associated with a database object (equivalent to granting ALTER, DELETE, INSERT, REFERENCES, SELECT, and UPDATE)	See “GRANT statement” [*SQL Anywhere Server - SQL Reference*].
ALTER	Yes	Allows the user to alter a database object	See “Permissions inherited through group membership” on page 449.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Permission	Inherited through group membership		More information
CONNECT	No	Allows the user to connect to the database	See "Creating new users" on page 450.
CONSOLIDATE	No	Identifies a consolidated database in SQL Remote	See "CONSOLIDATE permission" [*SQL Remote*].
DELETE	Yes	Allows the user to delete a database object	See "GRANT statement" [*SQL Anywhere Server - SQL Reference*].
INSERT	Yes	Allows the user to insert data into a database object	See "GRANT statement" [*SQL Anywhere Server - SQL Reference*].
INTEGRATED LOGIN	No	Allows the user to connect to the database using an integrated login	See "Using Windows integrated logins" on page 108.
KERBEROS LOGIN	No	Allows the user to connect to the database using a Kerberos login	See "Kerberos authentication" on page 116.
PUBLISH	No	Identifies the publisher of a database in SQL Remote	See "PUBLISH permission" [*SQL Remote*].
REFERENCES	Yes	Allows the user to create indexes on a table, and create foreign keys that reference the table	See "GRANT statement" [*SQL Anywhere Server - SQL Reference*].
REMOTE	No	Identifies a remote database in SQL Remote and MobiLink	See "GRANT REMOTE DBA statement [MobiLink] [SQL Remote]" [*SQL Anywhere Server - SQL Reference*].
SELECT	Yes	Allows the user to query a database object	See "GRANT statement" [*SQL Anywhere Server - SQL Reference*].
UPDATE	Yes	Allows the user to update a database object	See "GRANT statement" [*SQL Anywhere Server - SQL Reference*].

See also: "GRANT statement" [*SQL Anywhere Server - SQL Reference*].

Negative permissions

SQL Anywhere does not support negative permissions. This means that you cannot revoke a permission that was not explicitly granted.

For example, suppose user bob is a member of a group called sales. If a user grants DELETE permission on a table, T, to sales, then bob can delete rows from T. If you want to prevent bob from deleting from T, you cannot simply execute a REVOKE DELETE on T from bob, since the DELETE ON T permission was never granted directly to bob. In this case, you would have to revoke bob's membership in the sales group.

See:

- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "REVOKE statement" [*SQL Anywhere Server - SQL Reference*]

Authorities overview

In SQL Anywhere, **authorities** can be thought of as database-level permissions. As such, they are not necessarily associated with any object within the database (other than the user). For example, a user with BACKUP authority is allowed to back up the database. Authorities can also include database object permissions. For example, a user with PROFILE authority can perform application profiling and database tracing tasks, which involves using system tables and system procedures that are not available to other users (other than users with DBA authority).

BACKUP authority

The BACKUP authority allows a user to back up databases and transaction logs with archive or image backups using the BACKUP statement or dbbackup utility. BACKUP authority is not inherited through group membership, and can be granted only by a user with DBA authority. See "BACKUP statement" [*SQL Anywhere Server - SQL Reference*] and "Backup utility (dbbackup)" on page 767.

DBA authority

When you create a database, a single usable user ID is also created. By default, the first user ID is **DBA**, and the password is initially **sql** (passwords are case sensitive). You can change the name and password of the DBA user using the DBA USER and DBA PASSWORD clauses of the CREATE DATABASE statement or by specifying the dbinit -dba option. See "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*], and "Initialization utility (dbinit)" on page 799.

The DBA user ID automatically has DBA authority within the database. This level of permission enables DBA users to perform any activity in the database. They can create tables, change table structures, create new user IDs, revoke permissions from users, back up the database, and so on.

DBA authority is not inherited by group membership.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You only have DBA authority on a database if you are connected to it.

Users with DBA authority

A user with DBA authority becomes the database administrator. References made to the database administrator, or DBA, include any user or users with DBA authority.

Although DBA authority may be granted or transferred to other user IDs, this section assumes that the DBA user ID is the database administrator, and that the abbreviation DBA means both the DBA user ID and any user ID with DBA authority.

Adding new users

The DBA has the authority to add new users to the database. As the DBA adds users, they are also granted permissions to perform tasks on the database. Some users may need to simply look at the database information using SQL queries, others may need to add information to the database, and others may need to modify the structure of the database itself. Although some of the responsibilities of the DBA may be handed over to other user IDs, the DBA is responsible for the overall management of the database by virtue of the DBA authority.

The DBA has authority to create database objects and assign ownership of these objects to other user IDs.

Caution
To prevent unauthorized access to your data, you should change the password for the DBA user (or change the DBA user and password) before deploying the database.

PROFILE authority

The PROFILE authority allows a user to perform the following profiling, tracing, and diagnostic operations:

- application profiling
- diagnostic tracing
- procedure profiling
- request logging (request log creation and analysis)
- run the Index Consultant

The PROFILE authority is not inheritable by group membership.

Users with PROFILE authority are automatically added to the diagnostics group.

Performing procedure profiling and request logging requires the user to use the sa_server_option system procedure for configuration. Access to this procedure is granted to users with PROFILE authority, but only for the options related to procedure profiling and request logging.

When performing application profiling and diagnostic tracing, users with PROFILE authority, but not DBA authority, cannot create a separate database for storing the profiling and tracing data unless they have specific permissions to unload a database. However, they can store the data in the same database or in another database to which they can already connect.

To grant this authority, the database must have been created by a SQL Anywhere 11 or later database server, or have been upgraded to a version 11 or later database using the Upgrade utility (dbupgrad), or the ALTER DATABASE UPGRADE statement. See "Upgrade utility (dbupgrad)" on page 879 and "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "Application profiling" [*SQL Anywhere Server - SQL Usage*]
- "Advanced application profiling using diagnostic tracing" [*SQL Anywhere Server - SQL Usage*]
- "Index Consultant" [*SQL Anywhere Server - SQL Usage*]
- "Procedure profiling using system procedures" [*SQL Anywhere Server - SQL Usage*]
- "Request logging" [*SQL Anywhere Server - SQL Usage*]
- "DBA authority" on page 444
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

READCLIENTFILE authority

The READCLIENTFILE authority allows a user to read files on the client computer, for example when loading data from a file on a client computer.

The READCLIENTFILE authority can be inherited through group membership.

See also

- "Accessing data on client computers" [*SQL Anywhere Server - SQL Usage*]
- "WRITECLIENTFILE authority" on page 447
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "READ_CLIENT_FILE function [String]" [*SQL Anywhere Server - SQL Reference*]

READFILE authority

The READFILE authority allows a user to use the OPENSTRING clause in a SELECT statement to read a file. Without READFILE authority, the user can still use the OPENSTRING clause to query a string or BLOB value, but not a file.

The READFILE authority can be inherited through group membership.

For more information about using the OPENSTRING clause in a SELECT statement, see "FROM clause" [*SQL Anywhere Server - SQL Reference*].

REMOTE DBA authority

The REMOTE DBA authority grants a limited set of DBA permissions to SQL Remote or MobiLink synchronization users. The remote DBA authority avoids having to grant full DBA authority, thereby avoiding security problems associated with distributing DBA user IDs and passwords.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For more information about using the REMOTE DBA authority, see "GRANT REMOTE DBA statement [MobiLink] [SQL Remote]" [*SQL Anywhere Server - SQL Reference*].

RESOURCE authority

The RESOURCE authority allows a user to create database objects, such as tables, views, stored procedures, and triggers. The RESOURCE authority is not inherited through group membership, and can be granted only by a user with DBA authority.

To create a trigger, a user needs both RESOURCE authority and ALTER permissions on the table to which the trigger applies.

DBA authority is required to create database objects with different owners.

VALIDATE authority

The VALIDATE authority allows a user to perform database, table, index, and checksum validation using the VALIDATE statement or dbvalid utility. The VALIDATE authority is not inherited through group membership, and can be granted only by a user with DBA authority.

See also

- "VALIDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "Validation utility (dbvalid)" on page 881

WRITECLIENTFILE authority

The WRITECLIENTFILE authority allows a user to write to files on a client computer, for example when using the UNLOAD TABLE statement to write data to a client computer.

The WRITECLIENTFILE authority can be inherited through group membership.

See also

- "Accessing data on client computers" [*SQL Anywhere Server - SQL Usage*]
- "READCLIENTFILE authority" on page 446
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "READ_CLIENT_FILE function [String]" [*SQL Anywhere Server - SQL Reference*]
- "WRITE_CLIENT_FILE function [String]" [*SQL Anywhere Server - SQL Reference*]
- "UNLOAD statement" [*SQL Anywhere Server - SQL Reference*]

Understanding permissions

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

In SQL Anywhere, permissions allow users to access, create, modify, and delete database objects (tables, views, procedures, and so on). For example, to select data from a table, the user must either own the table, or have SELECT permissions on it.

A user's permissions can be grouped into the following main categories:

- **Permissions explicitly set for the user or group** These are the permissions that are explicitly set for a user or group to control whether they can create, modify, execute, or delete database objects.
- **Permissions acquired through ownership of an object** These are the permissions acquired by virtue of creating a data base object. For example, if a user creates a table, their ownership allows them to modify or delete the object.
- **Permissions inherited through group membership** These are the permissions inherited from a group to which a user or group belongs.
- **Permissions on disabled objects** You can grant permissions on disabled objects. Permissions to disabled objects are stored in the database and become effective when the object is enabled.

Permissions explicitly set for the user or group

You can give a user permission to execute system procedures and functions by granting EXECUTE permission for that object.

For tables, views, and dbspaces, there are several distinct permissions you may grant to user IDs:

Permission	Description
ALTER	Permission to alter the structure of a table or create a trigger on a table.
CREATE ON	Permission to create database objects on the specified dbspace.
DELETE	Permission to delete rows from a table or view.
INSERT	Permission to insert rows into a table or view.
REFERENCES	Permission to create indexes on a table and to create foreign keys that reference a table.
SELECT	Permission to look at information in a table or view.
UPDATE	Permission to update rows in a table or view. This can also granted on a set of columns in a table or view.
ALL	All the above permissions.

For more information about the permissions you can set on database objects, see "GRANT statement" [*SQL Anywhere Server - SQL Reference*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Permissions acquired through ownership of an object

A user who creates a new object within the database is called the **owner** of that object, and automatically has permission to perform any operation on that object. The owner of a table may modify the structure of that table, for instance, or may grant permissions to other database users to update the information within the table.

Users with DBA authority have permission to modify any component within the database, and so could delete a table created by another user. They have all the permissions regarding database objects that the owners of each object have. As well, users with DBA authority can also create database objects for other users. In this case, the owner of an object is not the user ID that executed the CREATE statement. Despite this possibility, the owner and creator of database objects as the same user are referred to interchangeably.

See also

- "Groups without passwords" on page 468

Permissions inherited through group membership

Setting permissions individually for each user of a database can be a time-consuming and error-prone process. For most databases, permission management based on groups, rather than on individual user IDs, is a much more efficient approach.

Each user ID can be a member of multiple groups, and they inherit all permissions from each of the groups.

For example, you may create groups for different departments in a company database (sales, marketing, and so on) and assign these groups permissions. Each salesperson becomes a member of the sales group, and automatically gains access to the appropriate areas of the database.

Managing user permissions and authorities

This section describes how to create new users and grant permissions and authorities to them. For most databases, the bulk of permission management should be performed using **groups**, rather than by assigning permissions to individual users one at a time. However, as a group is simply a user ID with special properties, you should read and understand this section before moving on to the discussion of managing groups.

Setting up individual user IDs

Even if there are no security concerns regarding a multi-user database, there are good reasons for setting up an individual user ID for each user. In addition to granting permissions to individual users, you can also grant permissions to groups of users. The administrative overhead is very low if a group with the appropriate permissions is set up.

You may want to use individual user IDs since:

- The Log Translation utility (dblog) can selectively extract the changes made by individual users from a transaction log. This is very useful when troubleshooting or piecing together what happened if data is incorrect.
- Sybase Central displays much more useful information so you can tell which connections belong to which users.
- Row locking messages (with the blocking option set to Off) are more informative.

Creating new users

You can create new users in both Sybase Central and Interactive SQL. In Sybase Central, you manage users or groups in the **Users & Groups** folder. In Interactive SQL, you can add a new user using the CREATE USER statement. For both tools, you need DBA authority to create new users.

All new users are automatically added to the PUBLIC group. Once you have created a new user, you can:

- add the user to other groups. See "Granting group membership to existing users or groups" on page 465.
- set the user's permissions on tables, views, and procedures. See "Managing user permissions and authorities" on page 449.
- set the user as the publisher or as a remote user of the database. See "User permissions" [*SQL Remote*].
- assign a login policy to the user. By default, a user is assigned to the root login policy. However, you can create and assign custom login policies. See "Managing login policies" on page 471.

Initial permissions for new users

By default, the permissions assigned to new users include:

- the ability to connect to the database (assuming a password has been specified for the user)
- the ability to view the data stored in the system views
- the ability to execute most system stored procedures

To access tables in the database, new users need to be assigned permissions.

A user with DBA authority can set the permissions granted automatically to new users by assigning permissions to the special PUBLIC user group. See "Special groups and users" on page 469.

Restrictions on user IDs and passwords

User IDs cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Passwords are case sensitive and they cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons
- be longer than 255 bytes in length

See "Setting a password" on page 451.

Create users

To create a new user (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Right-click **Users & Groups** and then choose **New** » **User**.
3. Follow the instructions in the **Create User Wizard**.

To create a new user (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a CREATE USER statement.

Example

This example adds a new user to the database with the user ID of M_Haneef and a password of Welcome.

```
CREATE USER M_Haneef
IDENTIFIED BY Welcome;
```

See also

- "CREATE USER statement" [*SQL Anywhere Server - SQL Reference*]

Setting a password

A user must have a password to be able to connect to the database. Passwords are case sensitive and they cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons
- be longer than 255 bytes in length

When passwords are created or changed, they are converted to UTF-8 before being hashed and stored in the database. If the database is unloaded and reloaded into a database with a different character set,

existing passwords continue to work. If the server cannot convert from the client's character set to UTF-8, then it is recommended that the password be composed of 7-bit ASCII characters as other characters may not work correctly.

Changing a password

To change a user password (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Click **Users & Groups**.
3. In the **Users & Groups** list, right-click a user and then choose **Properties**.
4. Select **This User Has A Password**.
5. Complete the **Password** and **Confirm Password** fields.
6. Click **Apply**.
7. Click **OK**.

To change a user password (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute an ALTER USER statement.

Example

```
ALTER USER M_Haneef
IDENTIFIED BY welcome;
```

Changing the DBA password

The default password for the DBA user for all databases is sql. You should change this password to prevent unauthorized access to your database.

To change the DBA password (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Click **Users & Groups**.
3. In the **Users & Groups** list, right-click **DBA** and then choose **Properties**.
4. Select **This User Has A Password**.
5. Complete the **Password** and **Confirm Password** fields.
6. Click **Apply**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

7. Click **OK**.

To change the DBA password (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a ALTER USER statement.

Example

The following command changes the password for the DBA user to welcome_DBA:

```
ALTER USER DBA
IDENTIFIED BY welcome_DBA;
```

See also

- "ALTER USER statement" [*SQL Anywhere Server - SQL Reference*]

Setting user and group options

In Sybase Central, configurable options for users and groups are located in the **User Options** and **Group Options** windows (the same window as for setting database options). In Interactive SQL, you can specify an option in a SET OPTION statement.

To set options for a user or group (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Click **Users & Groups**.
3. Right-click a user or group and then choose **Options**.
4. In the **Options** list, click an option.
5. Click **Set Permanent Now**.
6. Click **Close**.

To set the options for a user or group (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a SET OPTION statement.

See also

- "Set properties for database objects" [*SQL Anywhere Server - SQL Usage*]
- "Database options" on page 493

Granting authorities

You grant authorities in much the same way as you grant permissions.

To grant an authority to a user ID (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a GRANT statement, specifying the authorities you are granting, and the user-id of the recipient.

For example, to grant DBA authority, the appropriate SQL statement is:

```
GRANT DBA TO user-id;
```

For more information about the supported authorities in SQL Anywhere, see "Authorities overview" on page 444.

Granting permissions on tables

You can assign a set of permissions on individual tables and grant users combinations of these permissions to define their access to a table.

You can use either Sybase Central or Interactive SQL to set permissions. In Interactive SQL, you can use the GRANT statement to grant the following permissions on tables:

- The ALTER permission allows a user to alter the structure of a table or to create triggers on a table. The REFERENCES permission allows a user to create indexes on a table and to create foreign keys. These permissions grant the authority to modify the database schema, and so will not be assigned to most users. These permissions do not apply to views.
- The DELETE, INSERT, and UPDATE permissions grant the authority to modify the data in a table.
- The SELECT permission grants authority to look at data in a table, but does not give permission to change it.
- The ALL permission grants all the above permissions.
- The REFERENCES, SELECT, and UPDATE permissions can be restricted to a set of columns in the table or view.

To grant permissions on tables or columns (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Click **Tables**.
3. Right-click a table and then choose **Properties**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

4. Click the **Permissions** tab and configure the permissions for the table:
   - Click **Grant**.
   - Double-click a user or group.
   - In the permissions table, click the fields beside the user or group to set specific permissions.
   - Select a user and click **Change** to set specific permissions for a columns.
   - Click **OK**.
   - To revoke all permissions, select a user or group and click **Revoke**.
5. Click **Apply**.

Tips
You can also assign permissions from the **User Properties** or **Group Properties** window. To assign permissions to multiple users or groups, use the **Table Properties** window. To assign permissions to multiple tables, use the **User Properties** window.

To grant permissions on tables or columns (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a GRANT statement to assign the permission.

   See "GRANT statement" [*SQL Anywhere Server - SQL Reference*].

Example 1

All table permissions are granted in a very similar fashion. You can grant permission to M_Haneef to delete rows from the table named sample_table as follows:

1. Connect to the database as a user with DBA authority, or as the owner of sample_table.
2. Execute the following SQL statement:

```
GRANT DELETE
ON sample_table
TO M_Haneef;
```

Example 2

You can grant permission to M_Haneef to update the column_1 and column_2 columns only in the table named sample_table as follows:

1. Connect to the database as a user with DBA authority, or as the owner of sample_table.
2. Execute the following SQL statement:

```
GRANT UPDATE ( column_1, column_2 )
ON sample_table
TO M_Haneef;
```

Table permissions are limited in that they generally apply to all the data in a table, although the REFERENCES, SELECT, and UPDATE permissions can be granted to a subset of columns. You can fine-

tune user permissions by creating procedures that perform actions on tables, and then granting users the permission to execute the procedure.

See also

- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]

Granting permissions on views

Setting permissions on views is similar to setting them on tables.

For more information about the SQL statements involved, see "Granting permissions on tables" on page 454.

A user may perform an operation through a view if one or more of the following are true:

- The appropriate permission(s) on the view for the operation has been granted to the user by a user with DBA authority.
- The user has the appropriate permission(s) on all the base table(s) for the operation.
- The user was granted appropriate permission(s) for the operation on the view by a non-DBA user. This user must be either the owner of the view or have WITH GRANT OPTION of the appropriate permission(s) on the view. The owner of the view must be either:
  - a user with DBA authority.
  - a user that does not have DBA authority, but also the owner of all the base table(s) referred to by the view.
  - a user that does not have DBA authority, and is not the owner of some or all the base table(s) referred to by the view. However, the view owner has SELECT permission WITH GRANT OPTION on the base table(s) not owned and any other required permission(s) WITH GRANT OPTION on the base table(s) not owned for the operation.

    Instead of the owner having permission(s) WITH GRANT OPTION on the base table(s), permission(s) may have been granted to PUBLIC. This includes SELECT permission on system tables.

UPDATE permissions can be granted on an entire view or on individual columns within a view.

Note
You can grant permissions on disabled views. Permissions to disabled views are stored in the database and become effective when the object is enabled.

To grant permissions on views (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

2. Click **Views**.
3. Right-click a view and then choose **Properties**.
4. Click the **Permissions** tab.
5. Configure the permissions for the view:
   - Click **Grant**.
   - Double-click a user or group.
   - In the permissions table, click the fields beside the user or group to set specific permissions.
   - To revoke all permissions, select a user or group and click **Revoke**.
6. Click **Apply**.

Tip
You can also assign permissions from the **User Properties** or **Group Properties** window. Use the **View Properties** window to assign permissions to multiple users or groups. Use the **User Properties** or **Group Properties** window to assign permissions to multiple views.

See also

- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]

Granting users the right to grant permissions

You can assign each of the table and view permissions described with the WITH GRANT OPTION. This option gives the right to pass on the permission to other users.

In Sybase Central, you can specify a grant option by displaying the properties window of a user, group, or table, clicking the **Permissions** tab, and double-clicking the fields provided so that a check mark with two '+' signs appears.

Note
You can only specify WITH GRANT OPTION for users. Members of groups do not inherit the WITH GRANT OPTION if it is granted to a group.

Example

You can grant permission to M_Haneef to delete rows from the table named sample_table, and the right to pass on this permission to other users, as follows:

1. Connect to the database as a user with DBA authority, or as the owner of sample_table.
2. Execute the SQL statement:

```
GRANT DELETE ON sample_table
TO M_Haneef
WITH GRANT OPTION;
```

See "GRANT statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "Granting permissions on tables" on page 454
- "Permissions and authorities of groups" on page 467

Granting permissions on procedures

A user with DBA authority or the owner of the procedure may grant permission to execute stored procedures. The EXECUTE permission is the only permission that may be granted on a procedure.

The method for granting permissions to execute a procedure is similar to that for granting permissions on tables and views. However, the WITH GRANT OPTION clause of the GRANT statement does not apply to the granting of permissions on procedures.

You can use either Sybase Central or Interactive SQL to set permissions.

To grant permissions on procedures (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Click **Procedures & Functions**.
3. Right-click a procedure and then choose **Properties**.
4. Click the **Permissions** tab.
5. Configure the permissions for the procedure:
   - Click **Grant**.
   - Double-click a user or group.
   - To allow or revoke permission to execute a procedure, select a user or group and click the **Execute** column. A checkmark indicates the user or group can execute the procedure.
   - To revoke all permissions, select a user or group and click **Revoke**.
6. Click **Apply**.

Tip
You can also assign permissions from the **User Properties** or **Group Properties** window. Use the **Procedure Properties** window to assign permissions to multiple users or groups. Use the **User Properties** or **Group Properties** window to assign permissions to multiple procedures.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To grant permissions on procedures (SQL)

1. Connect to the database as a user with DBA authority or as the owner of the procedure.
2. Execute a GRANT EXECUTE ON statement.

Example

You can grant M_Haneef permission to execute a procedure named my_procedure, as follows:

1. Connect to the database as a user with DBA authority or as owner of my_procedure procedure.
2. Execute the SQL statement:

```
GRANT EXECUTE
ON my_procedure
TO M_Haneef;
```

Execution permissions of procedures

Procedures execute with the permissions of their owner. Any procedure that updates information in a table executes successfully only if the owner of the procedure has UPDATE permissions on the table.

As long as the procedure owner has the proper permissions, the procedure executes successfully when called by any user assigned permission to execute it, whether they have permissions on the underlying table. You can use procedures to allow users to perform well-defined activities on a table, without having any general permissions on the table.

See also

- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "Granting permissions on tables" on page 454

Execution permissions of triggers

The server executes triggers in response to a user action. Triggers do not require permissions to be executed. When a trigger executes, it does so with the permissions of the creator of the table with which it is associated.

See "Trigger execution permissions" [*SQL Anywhere Server - SQL Usage*].

Granting and revoking remote permissions

In Sybase Central, you can manage the remote permissions of both users and groups. Remote permissions allow normal users and groups to become remote users in a SQL Remote replication setup to exchange replication messages with the publishing database.

Granting remote permissions

You cannot grant remote permissions to a user until you define at least one message type in the database.

To grant remote permissions to a group, you must explicitly grant remote permissions to each user in the group. Remote permissions are not inherited by members of a group.

After granting a user remote permissions, you can subscribe the user to publications.

See "Introducing SQL Remote" [*SQL Remote*].

To grant remote permissions to users (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Click **Users & Groups**.
3. Right-click a user and then choose **Change To Remote User**.
4. Complete the fields and click **OK**.

Revoking remote permissions

Revoking remote permissions reverts a remote user to a normal user. Revoking these permissions also automatically unsubscribes that user from all publications.

To revoke remote permissions from remote users

1. Connect to the database as a user with DBA authority.
2. Click **Users & Groups** or **SQL Remote Users**.
3. Right-click a user and then choose **Revoke Remote**.
4. Click **Yes**.

Revoking user permissions and authorities

A user's permissions are a combination of those that have been granted and those that have been revoked. By revoking and granting permissions, you can manage the pattern of user permissions on a database.

A user with DBA authority or the owner of the procedure must issue this command.

If you are revoking connect permissions or table permissions from another user, the other user must not be connected to the database. You cannot revoke connect permissions from dbo.

The REVOKE statement revokes permissions that have been explicitly granted to the user (that is, not inherited from the groups to which they belong). The syntax for the REVOKE statement is the same as for the GRANT statement. For example, to revoke user M_Haneef's ability to execute my_procedure, the command is:

```
REVOKE EXECUTE
ON my_procedure
FROM M_Haneef;
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To revoke their permission to delete rows from sample_table, the command is:

```
REVOKE DELETE
ON sample_table
FROM M_Haneef;
```

When you add a user to a group, the user inherits all the permissions and inheritable authorities assigned to that group. SQL Anywhere does not allow you to revoke a subset of the permissions and authorities that a user inherits as a member of a group. You can only revoke permissions that are explicitly given by a GRANT statement. If you need to remove inherited permissions or authorities from a user, consider creating a new group with the required permissions and authorities, and making the user a member, or remove the user from the group and explicitly grant the permissions they require.

See also

- "REVOKE statement" [*SQL Anywhere Server - SQL Reference*]
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]

Deleting users from the database

You can delete a user from the database using both Sybase Central and Interactive SQL. The user being removed cannot be connected to the database during this procedure.

Deleting a user also deletes all database objects (such as tables) that they own, as well as any external logins for the user.

Only a user with DBA authority can delete a user.

To delete a user from the database (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Click **Users & Groups**.
3. Right-click a user and then choose **Delete**.
4. Click **Yes**.

To delete a user from the database (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a DROP USER statement.

Example

Remove the user M_Haneef from the database.

```
DROP USER M_Haneef;
```

See also

- "DROP USER statement" [*SQL Anywhere Server - SQL Reference*]
- "REVOKE statement" [*SQL Anywhere Server - SQL Reference*]
- "Revoking user permissions and authorities" on page 460
- "Deleting groups from the database" on page 469

Managing connected users

If you are working with Sybase Central, you can keep track of all users connected to the database. You can view properties of these connected users, and you can disconnect them if you want.

To display a list of all users connected to a database (Sybase Central)

- Select the database in the left pane, and click the **Connections** tab in the right pane.

  This tab displays all users currently connected to the database, regardless of the application that they used to connect (Sybase Central, Interactive SQL, or a custom client application).

To inspect the properties of a user's connection to a database (Sybase Central)

1. Select the database in the left pane, and click the **Connections** tab in the right pane.
2. Right-click the user and then choose **Properties**.
3. Review the properties for the user and click **OK**.

Disconnecting from a database

Disconnecting the current user

To disconnect from a database (Sybase Central)

1. Select a database.
2. Choose **File** » **Disconnect**.

To disconnect from a database (SQL)

- Execute a DISCONNECT statement.

Disconnecting other users

To disconnect users from a database (Sybase Central)

1. Select the database in the left pane, and click the **Connections** tab in the right pane.
2. Right-click the user and then choose **Disconnect**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To disconnect other users from a database (SQL)

1. Connect to the database as a user with DBA authority.
2. Use the sa_conn_info system procedure to determine the connection ID of the user you want to disconnect.
3. Execute a DROP CONNECTION statement.

To disconnect users from a database (SQL Anywhere Console utility)

1. Connect to the database from the SQL Anywhere Console utility.
2. In the **User ID** column, right-click the user and choose **Disconnect**.

   You can configure the columns that appear in the SQL Anywhere Console in the **Options** window, which can be accessed by choosing **File** » **Options**.

Examples

The following statement shows how to use the DISCONNECT statement to disconnect the current connection, conn1, in Interactive SQL:

```
DISCONNECT conn1;
```

The following statement shows how to use DISCONNECT in embedded SQL:

```
EXEC SQL DISCONNECT :conn-name
```

The following statement drops connection number 4.

```
DROP CONNECTION 4;
```

See also

- "DISCONNECT statement [ESQL] [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "DROP CONNECTION statement" [*SQL Anywhere Server - SQL Reference*]
- "Using the SQL Anywhere Console utility" on page 759
- "SQL Anywhere Console utility (dbconsole)" on page 848

Managing groups

A group can be thought of as a user ID with special permissions, such as the ability to have members. You grant and revoke permissions and authorities for a group in exactly the same manner as you do for users.

You can construct a hierarchy of groups where each group is a member of another group. Members, whether they be users or groups, inherit the authorities and permissions from its parent group. A user ID may belong to more than one group; the user-to-group relationship is many-to-many.

Just as with users, you can grant or revoke group permissions on a table, view, or procedure. When you do so, all members of the group inherit the change.

You can create a group without a password. This enables you to prevent users from connecting to the database using the group user ID. See "Groups without passwords" on page 468.

To administer authorities and permissions for a group, follow the same procedures that you do for administering permissions and authorities for users. See "Managing user permissions and authorities" on page 449.

To administer remote permissions for groups, see "Granting and revoking remote permissions" on page 459.

Special inheritance notes for groups

With the exception of the grant permission (GRANT ... WITH GRANT OPTION statement), users and groups inherit all permissions of the groups they are members of.

Members of a group can inherit only the following authorities set for the group they belong to.

- READCLIENTFILE
- READFILE
- WRITECLIENTFILE

Brief example

In the following example, two groups, group1 and group 2, are created. A user, bobsmith, is created and given membership in both groups. A table, table1, is created and group2 is given SELECT and INSERT permissions on the new table.

```
GRANT CONNECT, GROUP TO group1;
GRANT CONNECT, GROUP TO group2;
GRANT CONNECT TO bobsmith IDENTIFIED BY sql;
GRANT MEMBERSHIP IN GROUP group1 TO bobsmith;
GRANT MEMBERSHIP IN GROUP group2 TO bobsmith;

CREATE TABLE DBA.table1( column1 INT, modified_by VARCHAR(128) DEFAULT
USER );
GRANT SELECT, INSERT ON DBA.table1 TO group2;
```

Because bobsmith is a member of group2, he inherits select and insert permissions on table1 and can insert values into it as shown below:

```
CONNECT USER bobsmith IDENTIFIED BY sql;
INSERT INTO DBA.table1(column1) VALUES(1);
```

Creating groups

To create a new group (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Right-click **Users & Groups** and then choose **New** » **Group**.
3. Follow the instructions in the **Create Group Wizard**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To create a new group (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a GRANT GROUP TO statement. If the user ID you specify in this statement has not been created, the statement fails.

Example

Create the user ID personnel.

```
CREATE USER personnel
IDENTIFIED BY group_password;
```

Make the user ID personnel a group.

```
GRANT GROUP TO personnel;
```

See also

- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE USER statement" [*SQL Anywhere Server - SQL Reference*]
- "Creating new users" on page 450

Granting group membership to existing users or groups

You can add existing users to groups or add groups to other groups in both Sybase Central and Interactive SQL. In Sybase Central, you can control group membership in the right pane of users or groups. In Interactive SQL, you can make a user a member of a group with the GRANT statement.

When you assign a user membership in a group, they inherit all the permissions on tables, views, and procedures associated with that group. They also inherit any inheritable authorities.

Only a user with DBA authority can grant membership in a group.

To add a user or group to another group (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Click **Users & Groups**.
3. Double-click a user or group.
4. For a user:
   - From the **File** menu, choose **New** » **Memberships**.

   For a group:
   - Click the **Memberships** tab.
   - From the **File** menu, choose **New** » **Memberships**.

5. In the **Name** list, double-click a group.

   The new group appears on the **Memberships** tab in the right pane.

To add a user or group to another group (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a GRANT MEMBERSHIP IN GROUP statement, specifying the group and the users involved.

Example

Grant the user M_Haneef membership in the personnel group:

```
GRANT MEMBERSHIP
IN GROUP personnel
TO M_Haneef;
```

See also

- "Database permissions and authorities" on page 441
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "Creating new users" on page 450

Revoking group membership

You can remove users or groups from a group in both Sybase Central and Interactive SQL.

Removing a user or group from a group does not delete them from the database (or from other groups). To do this, you must delete the user/group itself.

Only a user with DBA authority can revoke membership in a group.

When you add a user to a group, the user inherits all the permissions assigned to that group. SQL Anywhere does not allow you to revoke a subset of the permissions that a user inherits as a member of a group because you can only revoke permissions that are explicitly given by a GRANT statement. If you need to have different permissions for different users, you can create different groups with the appropriate permissions, or you can explicitly grant each user the permissions they require.

To remove a user or group from another group (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Click **Users & Groups**.
3. Double-click a user or group.
4. In the **Memberships** pane, right-click a group and then choose **Remove Memberships**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Tip
You can also remove a user by double-clicking the group, clicking the **Members** tab in the right pane, right-clicking the user or group and choosing **Remove Members**.

To remove a user or group from another group (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a REVOKE MEMBERSHIP IN GROUP statement, specifying the group and user name.

Example

Remove the user M_Haneef from the personnel group:

```
REVOKE MEMBERSHIP
IN GROUP personnel
FROM M_Haneef;
```

See also

- "REVOKE statement" [*SQL Anywhere Server - SQL Reference*]
- "Creating new users" on page 450
- "Deleting users from the database" on page 461
- "Deleting groups from the database" on page 469

Permissions and authorities of groups

You grant permissions to groups in exactly the same way as any other user ID. Permissions on tables, views, and procedures are inherited by members of the group, including other groups and their members.

Ownership of database objects is associated with a single user ID and is not inherited by group members. If the user ID personnel creates a table, then the personnel user ID is the owner of that table and has the authority to make any changes to the table, and to grant privileges concerning the table to other users. Other user IDs who are members of personnel are not the owners of this table, and do not have these rights. Only granted permissions are inherited. For example, if a user with DBA authority or the personnel user ID explicitly grants SELECT permission on a table to the personnel user ID, all group members inherit select access to the table.

You can grant some authorities to groups as well.

Notes

Members of a group do not inherit the DBA, RESOURCE, and GROUP permissions. Even if the user ID has RESOURCE authority, the members of personnel do not have RESOURCE authority.

Note
You can only specify WITH GRANT OPTION for users. Members of groups do not inherit the WITH GRANT OPTION if it is granted to a group.

Referring to tables owned by groups

Groups are used for finding tables and procedures in the database. For example, the following query always finds the view SYS.SYSGROUPS, because all users belong to the PUBLIC group, and PUBLIC belongs to the SYS group which owns the SYSGROUPS view:

```
SELECT * FROM SYSGROUPS;
```

The SYSGROUPS view contains a list of *group-name*, *member-name* pairs representing the group memberships in your database.

If a table named employees is owned by the user ID personnel, and if **M_Haneef** is a member of the personnel group, then **M_Haneef** can refer to the employees table simply as employees in SQL statements. Users who are not members of the personnel group need to use the **qualified** name personnel.employees.

Creating a group to own the tables

A good practice to follow, that allows everyone to access the tables without qualifying names, is to create a group whose only purpose is to own the tables. Do not grant any permissions to this group, but make all users members of the group. You can then create permission groups and grant users membership in these permission groups as warranted.

If a user owns a table that has the same name as a table owned by a group, using the unqualified table name refers to the table owned by the user, not the one owned by the group. As well, if a user belongs to more than one group that has a table with the same name, the user must qualify the table name.

See "Database object names and prefixes" on page 479.

Groups without passwords

Users connected to a group's user ID have certain permissions. A user belonging to a group would have ownership permissions over any tables in the database created in the name of the group's user ID.

It is possible to set up a database so that only the DBA handles groups and their database objects, rather than permitting other user IDs to make changes to group membership. You can do this by disallowing connection as the group's user ID when creating the group. To do this, enter the CREATE USER statement without a password. The following statement creates a user ID personnel:

```
CREATE USER personnel;
```

This user ID can be granted group permissions, and other user IDs can be granted membership in the group, inheriting any permissions that have been given to personnel. However, nobody can connect to the database using the personnel user ID because it has no valid password.

The user ID personnel can be an owner of database objects, even though no user can connect to the database using this user ID. The CREATE TABLE statement, CREATE PROCEDURE statement, and CREATE VIEW statement all allow the owner of the object to be specified as a user other than that executing the statement. Only a user with DBA authority can perform this assignment of ownership.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Special groups and users

When you create a database, the SYS, PUBLIC, and dbo groups are also automatically created. None of these groups has passwords, so it is not possible to connect to the database as SYS, PUBLIC, or dbo. However, these groups serve important functions in the database.

- **SYS group** The SYS group owns the system tables and views for the database, which contain the full description of database schema, including all database objects and all user IDs.

  For more information about the system tables and views, together with a description of access to the tables, see "Tables" [*SQL Anywhere Server - SQL Reference*] and "System views" [*SQL Anywhere Server - SQL Reference*].

- **PUBLIC group** The PUBLIC group has SELECT permission on the system tables. As well, the PUBLIC group is a member of the SYS group, and has read access for some of the system tables and views, so any user of the database can find out information about the database schema. If you want to restrict this access, you can REVOKE PUBLIC's membership in the SYS group.

  Any new user ID is automatically a member of the PUBLIC group and inherits any permissions specifically granted to that group by a user with DBA authority. You can also REVOKE membership in PUBLIC for users if you want.

- **DIAGNOSTICS group** The DIAGNOSTIC group owns the diagnostic tables and views. Members in this group inherit permissions to run diagnostic operations. See "Monitoring and improving database performance" [*SQL Anywhere Server - SQL Usage*].

- **dbo group** The dbo group owns many system stored procedures and views. The dbo group is a member of the SYS group. The PUBLIC group is a member of the dbo group. The dbo group also owns tables used for UltraLite and MobiLink.

- **SA_DEBUG group** Membership in the SA_DEBUG group allows users to use the **SQL Anywhere Debugger**. See "Debugging procedures, functions, triggers, and events" [*SQL Anywhere Server - SQL Usage*].

- **SYS_SPATIAL_ADMIN_ROLE group** Membership in this group allows users to create, alter, or drop spatial reference systems and spatial units of measure. See "Introduction to spatial data" [*SQL Anywhere Server - Spatial Data Support*].

- **EXTENV_MAIN and EXTENV_WORKER users** EXTENV_MAIN and EXTENV_WORKER are used for external environment support. They are for internal use only and have limited permissions. See "SQL Anywhere external environment support" [*SQL Anywhere Server - Programming*].

Deleting groups from the database

You can delete a group from the database using both Sybase Central and Interactive SQL.

Deleting users or groups from the database is different from removing them from other groups. Deleting a group from the database does not delete its members from the database, although they lose membership in the deleted group.

Only a user with DBA authority can delete a group.

To delete a group from the database (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Click **Users & Groups**.
3. Right-click the group and then choose **Delete**.
4. Click **Yes**.

To delete a group from the database (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a REVOKE CONNECT FROM statement.

Example

Remove the group personnel from the database.

```
REVOKE CONNECT FROM personnel;
```

See also

- "REVOKE statement" [*SQL Anywhere Server - SQL Reference*]
- "Revoking user permissions and authorities" on page 460
- "Deleting users from the database" on page 461

Managing the resources connections use

Building a set of users and groups allows you to manage permissions on a database. Another aspect of database security and management is to limit the resources an individual user can use.

For example, you may want to prevent a single connection from taking too much of the available memory or CPU resources, so you can avoid having a connection slow down other users of the database.

SQL Anywhere provides a set of database options that users with DBA authority can use to control resources. These options are called **resource governors**.

Setting options

You can set database options using the SET OPTION statement, with the following syntax:

SET [**TEMPORARY**] **OPTION** ... [*userid*. | **PUBLIC**.]*option-name* = [*option-value*]

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Resources that can be managed

You can use the following options to manage resources:

- **max_cursor_count** Limits the number of cursors for a connection. See "max_cursor_count option" on page 554.
- **max_statement_count** Limits the number of prepared statements for a connection. See "max_statement_count option" on page 557.
- **priority** Sets the priority level at which requests from a connection are executed. See "priority option" on page 575.
- **max_priority** Controls the maximum priority level for connections. See "max_priority option" on page 555.

Database option settings are not inherited through the group structure.

See also

- "Database options" on page 493
- "SET OPTION statement" [*SQL Anywhere Server - SQL Reference*]

Managing login policies

A **login policy** is a named object in a database that consists of a set of rules that are applied when you create a database connection for a user. All new databases include a root login policy. You can modify the root login policy values, but you cannot delete the policy. Login policies govern only the rules for user login and are separate from authorities and permissions. Login policies are not inherited through group memberships.

The following table lists the settings that are governed by a login policy and includes the default settings for the root login policy:

Policy-option-name	Description	Default value	Applies to
password_life_time	The maximum number of days before a password must be changed.	Unlimited	All users including those with DBA authority
password_grace_time	The number of days before the password expires during which login is allowed, but the default post_login procedure issues warnings.	0	All users including those with DBA authority

Policy-option-name	Description	Default value	Applies to
password_expiry_on_next_login	If the value for this option is ON, the user's password will expire after the next login.	OFF	All users including those with DBA authority
locked	If the value for this option is ON, users are not allowed to establish new connections. Users with DBA authority cannot be locked. The reason_locked column of the sa_get_user_status system procedure returns a string generated by the database server that shows why a user is locked.	OFF	Users without DBA authority
max_connections	The maximum number of concurrent connections allowed for a user.	Unlimited	Users without DBA authority
max_failed_login_attempts	The maximum number of failed attempts, since the last successful attempt, to login before the user is locked.	Unlimited	Users without DBA authority
max_days_since_login	The maximum number of days that can elapse between two successive logins by the same user.	Unlimited	Users without DBA authority
max_non_dba_connections	The maximum number of concurrent connections that users without DBA authority can make. This option is only supported in the root login policy.	Unlimited	Users without DBA authority and only to the default login policy

A user is assigned the root login policy when:

- you create a new user and do not specify a login policy
- you use the Unload utility (dbunload) to rebuild a database created by a previous version of SQL Anywhere
- you upgrade a SQL Anywhere version 10 database using the Upgrade utility (dbupgrad) or the ALTER DATABASE UPGRADE statement

You can create, alter, and drop login policies. As well, you can create, alter, and drop users, and assign login policies to them. The sa_get_user_status system procedure lets you get information about the

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

current status of a user. See "sa_get_user_status system procedure" [*SQL Anywhere Server - SQL Reference*].

Inheritance of login policy settings

A default login policy called **root** is stored in the database and contains the default option values for all policies. If you want to use different settings than the defaults, you can either alter the root policy, or create a policy and then alter it to contain overrides for the defaults. A policy inherits its default settings from the root policy, unless it is altered to contain overrides.

For example, suppose the root policy value for max_connections is 5. You create a policy called myPolicy and alter it to set max_connections to Unlimited. Then, you create a user and assign the myPolicy login policy. When the user logs in, their login policy option settings are inherited from the root login policy with the exception of max_connections, which is set to Unlimited.

Inheritance of default values from the root policy is important to understand because if you subsequently change the value of an option setting in the root policy, you impact users of policies that rely on the default value for that setting. Similarly, if a root value is changed, it does not impact any users of policies that contain an override for that setting.

See also

- "CREATE LOGIN POLICY statement" [*SQL Anywhere Server - SQL Reference*]
- "ALTER LOGIN POLICY statement" [*SQL Anywhere Server - SQL Reference*]
- "DROP LOGIN POLICY statement" [*SQL Anywhere Server - SQL Reference*]

Modify the root login policy

To modify the root login policy (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, click **Login Policies**.
3. In the right pane, right-click **root** and choose **Properties**.
4. Modify a policy value and click **OK**.

To modify the root login policy (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute an ALTER LOGIN POLICY statement.

Example

This example creates overrides in the root login policy for the locked and max_connections values.

```
ALTER LOGIN POLICY root
locked=on
max_connections=5;
```

See also

- "ALTER LOGIN POLICY statement" [*SQL Anywhere Server - SQL Reference*]

Creating a new login policy

If you do not assign users to a login policy you create, they are assigned the root login policy.

To create a login policy (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Right-click **Login Policies** and then choose **New** » **Login Policy**.
3. Follow the instructions in the **Create Login Policy Wizard**.

To create a login policy (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a CREATE LOGIN POLICY statement. If you specify a login policy that already exists, the statement fails.

Example

This example creates the Test1 login policy with option values.

```
CREATE LOGIN POLICY Test1;
```

See also

- "CREATE LOGIN POLICY statement" [*SQL Anywhere Server - SQL Reference*]
- "Assigning a login policy to an existing user" on page 475
- "Altering a login policy" on page 476

Creating a user and assigning a login policy

If you create a user account and do not assign a login policy, they are assigned the root login policy.

To create a new user and assign a login policy (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Right-click **Users & Groups** and choose **New** » **User**.
3. Follow the instructions in the **Create User Wizard**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To create a new user and assign a login policy (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a CREATE USER statement.

Example

This example creates a user called SQLTester with the password "welcome" and assigns the Test1 login policy.

```
CREATE USER SQLTester IDENTIFIED BY welcome
LOGIN POLICY Test1;
```

See also

- "CREATE USER statement" [*SQL Anywhere Server - SQL Reference*]
- "Creating new users" on page 450

Assigning a login policy to an existing user

If you do not assign a customized login policy, users are assigned the root login policy. Use this procedure to change a user's login policy assignment.

To assign a login policy to an existing user (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, click **Users & Groups**.
3. In the right pane, right-click a user and then choose **Properties**.
4. In the **Login Policy** list, choose a login policy.
5. Click **OK**.

To assign a login policy to an existing user (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute an ALTER USER statement.

Example

This example assigns the Test2 login policy to SQLTester.

```
ALTER USER SQLTester
LOGIN POLICY Test2;
```

See also

- "ALTER USER statement" [*SQL Anywhere Server - SQL Reference*]
- "Creating new users" on page 450

Altering a login policy

To alter a login policy (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, click **Login Policies**.
3. In the right pane, right-click a login policy and choose **Properties**.
4. Alter the login policy value.
5. Click **OK**.

To alter a login policy (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute an ALTER LOGIN POLICY statement.

Example

This example creates overrides in the Test1 login policy for the locked and max_connections values.

```
ALTER LOGIN POLICY Test1
locked=on
max_connections=5;
```

See also

- "ALTER LOGIN POLICY statement" [*SQL Anywhere Server - SQL Reference*]
- "Assigning a login policy to an existing user" on page 475

Dropping a login policy

You cannot drop the root login policy. You must assign users to another login policy before dropping a customized login policy.

Note
You cannot drop a login policy if it is still assigned to a user.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To drop a login policy (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, click **Login Policies**.
3. On the **Login Policies** pane, right-click a login policy and choose **Delete**.
4. Click **Yes**.

To drop a login policy (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a DROP LOGIN POLICY statement.

Example

This example drops the Test1 login policy.

```
DROP LOGIN POLICY Test1;
```

See also

- "DROP LOGIN POLICY statement" [*SQL Anywhere Server - SQL Reference*]

Managing login policies on read-only databases

When you start a database in a read-only mode, the login policies are based on the existing persistent state of the database. The effect of any login policies you assign is limited to the current session.

If login management is enabled on a database that you later start in read-only mode, the following restrictions apply:

- Login management by the server is based on the state of the database before it is started.
- Explicit statements that change the state of the database are denied and result in an error.
- The server continues to maintain dynamic information, such as failed_login_attempts and last_login_time, for each user. However, this information is maintained in transient memory and is lost when you shut down the database. The database returns to the same state it was in before you started it.
- If the account is locked by the existing login management policy, a user cannot log in. Also, the usual methods for changing a password during log in are unavailable.
- If the database is read-only because of its role as a mirror database in a high availability system, then the effect of any statement executed on the primary database is reflected in the mirror database. Also, the dynamic information collected on the primary server is sent to the mirror database and is merged in transient memory with the information collected for the mirror database.

- If the database is read-only because of its role as a mirror database in a high availability system or as a copy node in a read-only scale-out system, then the effect of any statement executed on the primary database is reflected in the read-only database. Also, the dynamic information collected on the primary server is sent to the read-only database and is merged in transient memory with the information collected for the read-only database.

Specifying the permissions required to execute file administration statements

The -gu database server option controls who can execute file administration statements. You can use this option to specify which users are able to execute certain administration tasks. See "-gu dbeng12/dbsrv12 server option" on page 198.

There are four levels of permission for the use of file administration statements:

-gu option	Effect	Applies to
all	Anyone can execute file administration statements	Any database including utility database
none	No one can execute file administration statements	Any database including utility database
DBA	Only users with DBA authority can execute file administration statements	Any database including utility database
utility_db	Only the users who can connect to the utility database can execute file administration statements	Only the utility database

Examples

To prevent the use of the file administration statements, start the database server using the none permission level of the -gu option. The following command starts a database server and names it TestSrv. It loads the *mytestdb.db* database, but prevents anyone from using that server to create or delete a database, or execute any other file administration statement regardless of their resource creation rights, or whether they can load and connect to the utility database.

```
dbsrv12 -n TestSrv -gu none c:\mytestdb.db
```

To permit only the users knowing the utility database password to execute file administration statements, start the server by running the following command.

```
dbsrv12 -n TestSrv -su secret -gu utility_db
```

The following command starts Interactive SQL as a client application, connects to the server named TestSrv, loads the utility database, and connects the user.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
dbisql -c "UID=DBA;PWD=secret;DBN=utility_db;Host=host1;Server=TestSrv"
```

Having executed the above command successfully, the user connects to the utility database, and can execute file administration statements.

Database object names and prefixes

The name of every database object is an identifier.

For information about the rules for valid identifiers, see "Identifiers" [*SQL Anywhere Server - SQL Reference*].

In queries and sample SQL statements throughout this book, database objects from the sample database are generally referred to using their simple name. For example:

```
SELECT *
FROM Employees;
```

Tables, procedures, and views all have an owner. The DBA user ID owns the tables in the sample database. In some circumstances, you must prefix the object name with the owner user ID, as in the following statement.

```
SELECT *
FROM DBA.Employees;
```

The Employees table reference is said to be **qualified**. In other circumstances it is enough to give the object name. This section describes when you need to use the owner prefix to identify tables, views and procedures, and when you do not.

When referring to a database object, you require a prefix unless:

- You are the owner of the database object.
- The database object is owned by a group ID of which you are a member.

Example

Consider the following example of a corporate database. The user ID company created all the tables, and since this user ID belongs to the database administrator, it therefore has DBA authority.

```
CREATE USER Company
IDENTIFIED BY secret;
GRANT DBA TO Company;
```

The company user ID created the tables in the database.

```
CONNECT USER company IDENTIFIED BY secret;
CREATE TABLE company.Customers ( ... );
CREATE TABLE company.Products ( ... );
CREATE TABLE company.Orders ( ... );
CREATE TABLE company.Invoices ( ... );
CREATE TABLE company.Employees ( ... );
CREATE TABLE company.Salaries ( ... );
```

Not everybody in the company should have access to all information. Consider two user IDs in the sales department, Joe and Sally, who should have access to the Customers, Products, and Orders tables. To do this, you create a Sales group.

```
CREATE USER Sally IDENTIFIED BY xxxxx;
CREATE USER Joe IDENTIFIED BY xxxxx;
CREATE USER Sales IDENTIFIED BY xxxxx;
GRANT GROUP TO Sales;
GRANT ALL ON Customers TO Sales;
GRANT ALL ON Orders TO Sales;
GRANT SELECT ON Products TO Sales;
GRANT MEMBERSHIP IN GROUP Sales TO Sally;
GRANT MEMBERSHIP IN GROUP Sales TO Joe;
```

Now Joe and Sally have permission to use these tables, but they still have to qualify their table references because the table owner is Company, and Sally and Joe are not members of the Company group:

```
SELECT *
FROM company.Customers;
```

To rectify the situation, make the Sales group a member of the Company group.

```
GRANT GROUP TO Company;
GRANT MEMBERSHIP IN GROUP Company TO Sales;
```

Now Joe and Sally, being members of the Sales group, are indirectly members of the Company group, and can reference their tables without qualifiers. The following command now works:

```
SELECT *
FROM Customers;
```

Note

Joe and Sally do not have any extra permissions because of their membership in the company group. The company group has not been explicitly granted any table permissions. (The company user ID has implicit permission to look at tables like Salaries because it created the tables and has DBA authority.) So, Joe and Sally still get an error executing either of these commands:

```
SELECT *
FROM Salaries;
SELECT *
FROM company.Salaries;
```

In either case, Joe and Sally do not have permission to look at the Salaries table.

Using views and procedures for extra security

For databases that require a high level of security, defining permissions directly on tables has limitations. Any permission granted to a user on a table applies to the whole table. There are many cases when users' permissions need to be shaped more precisely than on a table-by-table basis. For example:

- It is not desirable to give access to personal or sensitive information stored in an employee table to users who need access to other parts of the table.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- You may want to give sales representatives update permissions on a table containing descriptions of their sales calls, but limit such permissions to their own calls.

In these cases, you can use views and stored procedures to tailor permissions to suit the needs of your organization. This section describes some of the uses of views and procedures for permission management.

See also

- "Working with views" [*SQL Anywhere Server - SQL Usage*]
- "Granting permissions on views" on page 456

Using views for tailored security

Views are computed tables that contain a selection of rows and columns from base tables. Views are useful for security when it is appropriate to give a user access to just one portion of a table. The portion can be defined as rows or columns. For example, you may want to disallow a group of users from seeing the Salary column of an employee table, or you may want to limit a user to see only the rows of a table they have created.

Example 1

The Sales manager needs access to information in the database concerning employees in the department. However, there is no reason for the manager to have access to information about employees in other departments.

This example describes how to create a user ID for the sales manager, create views that provide the information she needs, and grant the appropriate permissions to the sales manager user ID.

1. Create the new user ID using the GRANT statement. While logged in as a user with DBA authority, execute the following statements:

   ```
   CONNECT DBA
   IDENTIFIED by sql;

   CREATE USER SalesManager
   IDENTIFIED BY sales;
   ```

2. Define a view that only looks at sales employees as follows:

   ```
   CREATE VIEW EmployeeSales AS
     SELECT EmployeeID, GivenName, Surname
     FROM Employees
     WHERE DepartmentID = 200;
   ```

   The table reference could be qualified with the owner to avoid an ambiguous reference to an identically named table.

3. Give SalesManager permission to look at the view:

   ```
   GRANT SELECT
   ON EmployeeSales
   TO SalesManager;
   ```

You use exactly the same command to grant permission on views and on tables.

Example 2

The next example creates a view which allows the Sales Manager to look at a summary of sales orders. This view requires information from more than one table for its definition:

1. Create the view.

```
CREATE VIEW OrderSummary AS
  SELECT OrderDate, Region, SalesRepresentative, CompanyName
  FROM SalesOrders
    KEY JOIN Customers;
```

2. Grant permission for the Sales Manager to examine this view.

```
GRANT SELECT
ON OrderSummary
TO SalesManager;
```

3. To check that the process has worked properly, connect to the SalesManager user ID and look at the views you created:

```
CONNECT SalesManager
IDENTIFIED BY sales;
SELECT *
FROM DBA.EmployeeSales;
SELECT *
FROM DBA.OrderSummary;
```

No permissions have been granted to the Sales Manager to look at the underlying tables. The following commands produce permission errors.

```
SELECT * FROM GROUPO.Employees;
SELECT * FROM GROUPO.SalesOrders;
```

Other permissions on views

The previous example shows how to use views to tailor SELECT permissions. You can grant INSERT, DELETE, and UPDATE permissions on views in the same way.

Using procedures for tailored security

While views restrict data access, procedures restrict the actions a user may take. A user can have EXECUTE permission on a procedure without having any permissions on the table or tables on which the procedure acts. See “Granting permissions on procedures” on page 458.

Strict security

For strict security, you can disallow all access to the underlying tables, and grant permissions to users or groups of users to execute certain stored procedures. This approach strictly defines how data in the database can be modified.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Changing ownership on nested objects

Views and procedures can access underlying objects that are owned by different users. For example, if usera, userb, userc, and userd were four different users, userd.viewd could be based on userc.viewc, which could be based on userb.viewb, which could be based on usera.tablea. Similarly for procedures, userd.procd could call userc.procc, which could call userb.procb, which could insert into usera.tablea.

The following Discretionary Access Control (DAC) rules apply to nested views and tables:

- To create a view, the user must have SELECT permission on all the base objects (for example tables and views) in the view.
- To access a view, the view owner must have been granted the appropriate permission on the underlying tables or views with the GRANT OPTION and the user must have been granted the appropriate permission on the view.
- Updating with a WHERE clause requires both SELECT and UPDATE permission.
- If a user owns the tables in a view definition, the user can access the tables through a view, even if the user is not the owner of the view and has not been granted access on the view.

The following DAC rules apply to nested procedures:

- A user does not require any permissions on the underlying objects (for example tables, views or procedures) to create a procedure.
- For a procedure to execute, the owner of the procedure needs the appropriate permissions on the objects that the procedure references.
- Even if a user owns all the tables referenced by a procedure, the user will not be able to execute the procedure to access the tables unless the user has been granted EXECUTE permission on the procedure.

Following are some examples that describe this behavior.

Example 1: User1 creates table1, and user2 creates view2 on table1

- User1 can always access table1, since user1 is the owner.
- User1 can always access table1 through view2, since user1 is the owner of the underlying table. This is true even if user2 does not grant permission on view2 to user1.
- User2 can access table1 directly or through view2 if user1 grants permission on table1 to user2.
- User3 can access table1 if user1 grants permission on table1 to user3.
- User3 can access table1 through view2 if user1 grants permission on table1 to user2 with grant option *and* user2 grants permission on view2 to user3.

Example 2: User2 creates procedure2 that accesses table1

- User1 can access table1 through procedure2 if user2 grants EXECUTE permission on procedure2 to user1. Note that this is different from the case of view2, where user1 did not need permission on view2.

Example 3: User1 creates table1, user2 creates table2, and user3 creates view3 joining table1 and table2

- User3 can access table1 and table2 through view3 if user1 grants permission on table1 to user3 *and* user2 grants permission on table2 to user3.
- If user3 has permission on table1 but not on table2, then user3 cannot use view3, even to access the subset of columns belonging to table1.
- User1 or user2 can use view3 if (a) user1 grants permission with grant option on table1 to user3, (b) user2 grants permission with grant option on table2 to user3, *and* (c) user3 grants permission on view3 to that user.

How user permissions are assessed

Groups do introduce complexities in the permissions of individual users. Suppose user M_Haneef has SELECT and UPDATE permissions on a specific table individually, but is also a member of two groups. Suppose one of these groups has no access to the table at all, and one has only SELECT access. What are the permissions in effect for this user?

SQL Anywhere decides whether a user ID has permission to perform a specific action in the following manner:

1. If the user ID has DBA authority, the user ID can perform any action in the database.
2. Otherwise, permission depends on the permissions assigned to the individual user. If the user ID has been granted permission to perform the action, then the action proceeds.
3. If no individual settings have been made for that user, permission depends on the permissions of each of the groups to which the member belongs. If any of these groups has permission to perform the action, the user ID has permission by virtue of membership in that group, and the action proceeds.

This approach minimizes problems associated with the order in which permissions are set.

Users and permissions in the catalog

The database system views contain information about the current users of a database and about their permissions.

The special user ID SYS owns the system views. You cannot connect using the SYS user ID.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Users with DBA authority have SELECT access to all system views, but not the underlying system tables. The access of other users to some tables and views is also limited. For example, only a user with DBA authority has access to the SYS.SYSUSERPERM view, which contains all information about the permissions of users of the database, and the encrypted passwords of each user ID. However, SYS.SYSUSERPERMS is a view containing all information in SYS.SYSUSERPERM except for the password, and by default all users have SELECT access to this view. You can fully modify all permissions and group memberships set up in a new database for SYS, PUBLIC, DBA, and dbo.

The following table summarizes the system views containing information about user IDs, groups, and permissions. The user ID SYS owns all the listed views, and so their qualified names are SYS.SYSUSERPERM and so on.

Appropriate SELECT queries on these views generate all the user ID and permission information stored in the database.

View	Default	Contents
SYSCOLAUTH	PUBLIC	Information from SYSCOLPERM in a more readable format. See "SYSCOLAUTH consolidated view" [*SQL Anywhere Server - SQL Reference*].
SYSCOLPERM	PUBLIC	All columns with SELECT or UPDATE permission given by the GRANT command. See "SYSCOLPERM system view" [*SQL Anywhere Server - SQL Reference*].
DUMMY	PUBLIC	Dummy table that can be used to find the current user ID. See "DUMMY system table" [*SQL Anywhere Server - SQL Reference*].
SYSGROUP	PUBLIC	One row for each member of each group. See "SYSGROUP system view" [*SQL Anywhere Server - SQL Reference*].
SYSGROUPS	PUBLIC	Information from SYSGROUP in a more readable format. See "SYSGROUPS consolidated view" [*SQL Anywhere Server - SQL Reference*].
SYSPROCAUTH	PUBLIC	Information from SYSPROCPERM in a more readable format. See "SYSPROCAUTH consolidated view" [*SQL Anywhere Server - SQL Reference*].
SYSPROCPERM	PUBLIC	Each row holds one user granted permission to use one procedure. See "SYSPROCPERM system view" [*SQL Anywhere Server - SQL Reference*].

View	Default	Contents
SYSTABAUTH	PUBLIC	Information from SYSTABLEPERM in a more readable format. See "SYSTABAUTH consolidated view" [*SQL Anywhere Server - SQL Reference*].
SYSTABLEPERM	PUBLIC	All permissions on table given by the GRANT commands. See "SYSTABLEPERM system view" [*SQL Anywhere Server - SQL Reference*].
SYSUSER	DBA only	Information on all users in the database. See "SYSUSER system view" [*SQL Anywhere Server - SQL Reference*].
SYSUSERAUTH	DBA only	All information in SYSUSERPERM except for user numbers. See "SYSUSERAUTH compatibility view (deprecated)" [*SQL Anywhere Server - SQL Reference*].
SYSUSERAUTHORITY	PUBLIC	Authority granted for each user ID. See "SYSUSERAUTHORITY system view" [*SQL Anywhere Server - SQL Reference*].
SYSUSERLIST	PUBLIC	All information in SYSUSERAUTH except for passwords. See "SYSUSERLIST compatibility view (deprecated)" [*SQL Anywhere Server - SQL Reference*].
SYSUSERPERM	DBA only	Database-level permissions and password for each user ID. See "SYSUSERPERM compatibility view (deprecated)" [*SQL Anywhere Server - SQL Reference*].
SYSUSERPERMS	PUBLIC	All information in SYSUSERPERM except for passwords. See "SYSUSERPERMS compatibility view (deprecated)" [*SQL Anywhere Server - SQL Reference*].

Database options

Database options control many aspects of database behavior. For example, you can use database options for the following purposes:

- **Compatibility** You can control how much like Adaptive Server Enterprise your SQL Anywhere database operates, and whether SQL that does not conform to SQL/2008 generates errors. See "Compatibility options" on page 500.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Error handling** You can control what happens when errors such as divide by zero errors occur.
- **Concurrency and transactions** You can control the degree of concurrency, and details of COMMIT behavior.

Setting database options

You set options with the SET OPTION statement. It has the following general syntax:

SET [**EXISTING**] [**TEMPORARY**] **OPTION**
[*userid*. | **PUBLIC**.]*option-name* = [*option-value*]

You can specify a user ID or group name to set the option for that user or group only. Every user belongs to the PUBLIC group. If no user ID or group is specified, the option change is applied to the currently logged on user ID that issued the SET OPTION statement.

Any option, whether user-defined or not, must have a public setting before a user-specific value can be assigned. The database server does not support setting TEMPORARY values for user-defined options.

For example, the following statement applies an option change to the user DBA, if DBA is the user that issues it:

```
SET OPTION blocking_timeout = 3;
```

The following statement applies a change to the PUBLIC user ID, a user group to which all users belong. You must have DBA authority to execute this statement.

```
SET OPTION PUBLIC.login_mode = 'Standard';
```

If *option-value* is omitted, the specified option setting is deleted from the database. If it was a personal option setting, the value reverts back to the PUBLIC setting. If a TEMPORARY option is deleted, the option setting reverts back to the permanent setting.

Caution
Changing option settings while fetching rows from a cursor is not supported because it can lead to unreliable results. For example, changing the date_format setting while fetching from a cursor would lead to different date formats among the rows in the result set. Do not change option settings while fetching rows.

Note
In databases that use a Turkish collation or are case sensitive, executing a query on SYSOPTION or a query like the following may not match any rows if the option name is used with the wrong case:

```
SELECT * FROM sa_conn_properties( ) WHERE propname = 'BLOCKING';
```

For information about the proper case for option names, see “Alphabetical list of options” on page 505.

Setting database options from Sybase Central

To set database options (Sybase Central)

1. Open the database server.
2. Right-click the database and choose **Options**.
3. Edit the values.

Tips

With the **Database Options** window, you can also set database options for specific users and groups (when you open this window for a user or group, it is called the **User Options** window or **Group Options** window, respectively).

When you set options for the database itself, you are actually setting options for the PUBLIC group in that database because all users and groups inherit option settings from PUBLIC.

See also

- "SET OPTION statement" [*SQL Anywhere Server - SQL Reference*]

Scope and duration of database options

You can set options at 3 levels of scope: public, user, and temporary.

Temporary options take precedence over user and public settings. User-level options take precedence over public settings. If you set a user level option for the current user, the corresponding temporary option is set as well.

Some options (such as COMMIT behavior) are database-wide in scope. Setting these options requires DBA authority. Other options (such as isolation_level) can also be applied to just the current connection, and need no special permissions.

Changes to option settings take place at different times, depending on the option. Changing a global option such as recovery_time takes place the next time the database is started.

Generally, only options that affect the current connection take place immediately. You can change option settings in the middle of a transaction, for example. One exception to this is that changing options when a cursor is open can lead to unreliable results. For example, changing date_format may not change the format for the next row when a cursor is opened. Depending on the way the cursor is being retrieved, it may take several rows before the change works its way to the user.

Setting public options

DBA authority is required to set an option for the PUBLIC user ID.

Changing the value of an option for the PUBLIC user ID sets the permanent value of the option for all users who have not SET their own value. An option value cannot be set for an individual user ID unless there is already a PUBLIC user ID setting for that option.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Some options which can only be set for the PUBLIC user take effect immediately for existing connections, even though the changed setting will not be visible to users via the CONNECTION_PROPERTY function. An example of this is the global_database_id option. For this reason, PUBLIC-only options should not be changed while other users are connected to the database.

Setting temporary options

Adding the TEMPORARY keyword to the SET OPTION statement changes the duration of the change. Ordinarily an option change is permanent. It does not change until it is explicitly changed using the SET OPTION statement.

When the SET TEMPORARY OPTION statement is executed, the new option value takes effect only for the current connection, and only for the duration of the connection. This new setting is held only in server memory and is not reflected in the SYSOPTION catalog view.

When the SET TEMPORARY OPTION is used to set a PUBLIC option, the change is in place for as long as the database is running. When the database is shut down, temporary options for the PUBLIC user ID revert back to their permanent value.

Setting a temporary option for the PUBLIC user ID offers a security advantage. For example, when the login_mode option is enabled, the database relies on the login security of the system on which it is running. Enabling it as a temporary option setting means that a database relying on the security of a Windows domain will not be compromised if the database is shut down and copied to a local computer. In this case, the login_mode option will revert to its permanent value, which could be Standard, a mode where integrated logins are not permitted.

Setting options for a SQL statement

The INSERT, UPDATE, DELETE, SELECT, UNION, EXCEPT, and INTERSECT statements have an OPTION clause that lets you specify how materialized views are used by the statement and how the query is optimized. This clause can also be used to specify an option setting that takes precedence over any public or temporary option settings that are in effect, for that statement only. You can change the setting of the following options in the OPTION clause:

- isolation_level
- max_query_tasks
- optimization_goal
- optimization_level
- optimization_workload
- user_estimates

See also

- "INSERT statement" [*SQL Anywhere Server - SQL Reference*]
- "UPDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "DELETE statement" [*SQL Anywhere Server - SQL Reference*]
- "SELECT statement" [*SQL Anywhere Server - SQL Reference*]
- "UNION statement" [*SQL Anywhere Server - SQL Reference*]
- "EXCEPT statement" [*SQL Anywhere Server - SQL Reference*]
- "INTERSECT statement" [*SQL Anywhere Server - SQL Reference*]
- "isolation_level option" on page 544
- "max_query_tasks option" on page 556
- "optimization_goal option" on page 567
- "optimization_level option" on page 568
- "optimization_workload option" on page 569
- "user_estimates option" on page 612

Finding option settings

You can obtain a list of option settings, or the values of individual options, in a variety of ways.

Getting a list of option values

- You can list all current system-defined option settings via the sa_conn_options system procedure.

  ```
  CALL sa_conn_options;
  ```

  To order this list alphabetically, you can execute the following statement:

  ```
  SELECT *
  FROM sa_conn_options( )
  ORDER BY OptionName;
  ```

  If you want to filter the result or order by anything other than name, you could also use a WHERE clause. For example:

  ```
  SELECT *
  FROM sa_conn_options( )
  WHERE OptionDescription LIKE '%date%'
  ORDER BY PropNum;
  ```

  See "sa_conn_options system procedure" [*SQL Anywhere Server - SQL Reference*].

- Current system-defined option settings for your connection are also available as a subset of **connection properties**. You can list all connection properties using the sa_conn_properties system procedure.

  ```
  CALL sa_conn_properties;
  ```

  To order this list alphabetically, you can execute the following statement:

  ```
  SELECT *
  FROM sa_conn_properties( )
  ORDER BY PropName;
  ```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See "sa_conn_properties system procedure" [*SQL Anywhere Server - SQL Reference*].

- The results of the sa_conn_options and sa_conn_properties system procedures include connection-level settings for system-defined options which override user or PUBLIC settings.

  For user-defined options, their values are not returned by either the sa_conn_options or sa_conn_properties system procedures. You can instead query the SYSOPTION or SYSOPTIONS catalog views directly in order to determine the value of a specific user-defined option. The following query on the SYSOPTIONS system view displays all public-level and user-level settings for all system- and user-defined options that have been permanently set:

  ```
  SELECT *
  FROM SYSOPTIONS;
  ```

  Temporarily setting a server-defined option for a specific connection using the SET TEMPORARY OPTION statement does not cause the SYSOPTION (or SYSOPTIONS) catalog view to be updated. Temporary changes to option settings at the connection level are held only in server memory. To determine the current setting for a system-defined option, you should use the sa_conn_options or sa_conn_properties system procedures, the CONNECTION_PROPERTY function, or (in embedded SQL) the GET OPTION statement.

- In Interactive SQL, the SET statement with no arguments lists the current setting of system-defined connection, database, and Interactive SQL options.

  ```
  SET;
  ```

- In Sybase Central, select a database, and then choose **File** » **Options**.

Getting individual option values

You can obtain the current setting for a single system-defined option by using the CONNECTION_PROPERTY system function. For example, the following statement reports the value of the ansi_blanks option:

```
SELECT CONNECTION_PROPERTY ( 'ansi_blanks' );
```

CONNECTION_PROPERTY cannot be used to obtain the values of user-defined options.

See "CONNECTION_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]

Within embedded SQL programs, you can use the GET OPTION statement to determine the value of an option (system- or user-defined) within your embedded SQL application. See "GET OPTION statement [ESQL]" [*SQL Anywhere Server - SQL Reference*].

See also

- "SYSOPTION system view" [*SQL Anywhere Server - SQL Reference*]
- "sa_conn_options system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_conn_options system procedure" [*SQL Anywhere Server - SQL Reference*]
- "CONNECTION_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]
- "GET OPTION statement [ESQL]" [*SQL Anywhere Server - SQL Reference*]
- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "SET statement [T-SQL]" [*SQL Anywhere Server - SQL Reference*]
- "SET REMOTE OPTION statement [SQL Remote]" [*SQL Anywhere Server - SQL Reference*]

Monitoring option settings

You can use the sa_server_option system procedure to instruct the database server to send a message or return an error when an attempt is made to set a database option.

You use the OptionWatchList property to create a list the options that you want to monitor, and the OptionWatchAction property to specify the action the database server should take when an attempt is made to set an option that is being monitored.

For example, the following command instructs the database server to monitor the database options automatic_timestamp, float_as_double, and tsql_hex_constant:

```
CALL dbo.sa_server_option(
'OptionWatchList','automatic_timestamp,float_as_double,tsql_hex_constant' );
```

The following command instructs the database server to return an error if an attempt is made to set an option specified in the OptionWatchList property:

```
CALL dbo.sa_server_option( 'OptionWatchAction','ERROR' );
```

See also

- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "OptionWatchAction database property" on page 671
- "OptionWatchList database property" on page 671

Initial option settings

Connections to SQL Anywhere can be made through the TDS protocol (Sybase Open Client and jConnect JDBC connections) or through the SQL Anywhere protocol (ODBC and embedded SQL).

If you have users who use both TDS and the SQL Anywhere-specific protocol, you can configure their initial settings using stored procedures. SQL Anywhere uses this method to set Sybase Open Client connections and jConnect connections to reflect default Adaptive Server Enterprise behavior.

The initial settings are controlled using the login_procedure option. This option names a stored procedure to run when users connect. The default setting is to use the sp_login_environment system procedure. You can change this behavior as necessary.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

In turn, sp_login_environment checks to see if the connection is being made over TDS. If it is, it calls the sp_tsql_environment procedure, which sets several options to new default values for the current connection.

See also

- “login_procedure option” on page 549
- “sp_login_environment system procedure” [*SQL Anywhere Server - SQL Reference*]
- “sp_tsql_environment system procedure” [*SQL Anywhere Server - SQL Reference*]

Deleting option settings

If *option-value* is omitted, the specified option setting is removed from the database. If it was a personal option setting, the value reverts back to the PUBLIC setting. If a TEMPORARY option is deleted, the option setting reverts back to the permanent setting.

For example, the following statement resets the ansi_blanks option to its default value:

```
SET OPTION ansi_blanks =;
```

See “SET OPTION statement” [*SQL Anywhere Server - SQL Reference*].

Option classification

SQL Anywhere provides many options. It is convenient to divide them into a few general classes. The classes of options are:

- “Database options” on page 493
- “Compatibility options” on page 500
- “Synchronization options” on page 503
- “SQL Remote options” on page 504
- “Interactive SQL options” on page 739

Database options

The following options allow you to configure the behavior of a SQL Anywhere database server.

Option	Values	Default
“allow_nulls_by_default option” on page 505	On, Off	On Off for Sybase Open Client and jConnect connections
“allow_read_client_file option” on page 506	On, Off	Off

Option	Values	Default
“allow_snapshot_isolation option” on page 506	On, Off	Off
“allow_write_client_file option” on page 508	On, Off	Off
“ansi_blanks option” on page 508	On, Off	Off
“ansi_close_cursors_on_rollback option” on page 509	On, Off	Off
“ansi_permissions option” on page 510	On, Off	On
“ansi_substring option” on page 511	On, Off	On
“ansi_update_constraints option” on page 512	Off, Cursors, Strict	Cursors
“ansinull option” on page 513	On, Off	On
“auditing option” on page 514	On, Off	Off
“auditing_options option” on page 515	Reserved	Reserved
“background_priority option [deprecated]” on page 515	On, Off	Off
“blocking option” on page 516	On, Off	On
“blocking_others_timeout option” on page 517	Integer, in milliseconds	0
“blocking_timeout option” on page 517	Integer, in milliseconds	0
“chained option” on page 518	On, Off	On Off for Sybase Open Client and jConnect connections
“checkpoint_time option” on page 519	Number of minutes	60
“cis_option option” on page 519	0, 7	0
“cis_rowset_size option” on page 520	Integer	50
“close_on_endtrans option” on page 520	On, Off	On Off for jConnect connections

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Values	Default
“collect_statistics_on_dml_updates option” on page 521	On, Off	On
“conn_auditing option” on page 522	On, Off	On
“connection_authentication option” on page 523	String	Empty string
“continue_after_raiserror option” on page 524	On, Off	On
“conversion_error option” on page 525	On, Off	On
“cooperative_commit_timeout option” on page 526	Integer, in milliseconds	250
“cooperative_commits option” on page 526	On, Off	On
“database_authentication” on page 527	String	Empty string
“date_format option” on page 528	String	YYYY-MM-DD
“date_order option” on page 530	MDY, YMD, DMY	YMD
“debug_messages option” on page 531	On, Off	Off
“dedicated_task option” on page 531	On, Off	Off
“default_dbspace option” on page 532	String	Empty string (use the system dbspace)
“default_timestamp_increment option” on page 533	Integer, in microseconds from 1 to 1000000	1
“delayed_commit_timeout option” on page 534	Integer, in milliseconds	500
“delayed_commits option” on page 534	On, Off	Off
“divide_by_zero_error option” on page 536	On, Off	On
“escape_character option” on page 536	Reserved	Reserved
“exclude_operators option” on page 537	Reserved	Reserved
“extended_join_syntax option” on page 537	On, Off	On
“fire_triggers option” on page 538	On, Off	On

Option	Values	Default
“first_day_of_week option” on page 538	1, 2, 3, 4, 5, 6, 7	7 (Sunday is the first day of the week)
“for_xml_null_treatment option” on page 539	Empty, Omit	Omit
“force_view_creation option” on page 540	Reserved	Reserved
“global_database_id option” on page 540	Integer	2147483647
“http_connection_pool_basesize option” on page 541	Integer (from 0 to 1000)	10
“http_connection_pool_timeout option” on page 542	Integer, in seconds from 1 to 86400	60
“http_session_timeout option” on page 542	Integer	30
“integrated_server_name option” on page 543	String	NULL
“isolation_level option” on page 544	0, 1, 2, 3, Snapshot, Statement-snapshot, Readonly-statement-snapshot	0
“java_location option” on page 545	String	Empty string
“java_vm_options option” on page 546	String	Empty string
“log_deadlocks option” on page 547	On, Off	Off
“login_mode option” on page 547	Standard, Integrated, Kerberos, Mixed (deprecated)	Standard
“login_procedure option” on page 549	String	sp_login_environment
“materialized_view_optimization option” on page 551	Disabled, Fresh, Stale, *N* { Minute[s] | Hour[s] | Day[s] | Week[s] | Month[s] }	Stale
“max_client_statements_cached option” on page 552	Integer	10
“max_cursor_count option” on page 554	Integer	50

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Values	Default
“max_plans_cached option” on page 554	Integer	20
“max_priority option” on page 555	Critical, High, Above Normal, Normal, Below normal, Low, Background	Normal
“max_query_tasks option” on page 556	Integer	0
“max_recursive_iterations option” on page 557	Integer	100
“max_statement_count option” on page 557	Integer >=0	50
“max_temp_space option” on page 559	Integer [k|m|g|p]	0
“min_password_length option” on page 560	Integer >=0	0 characters
“nearest_century option” on page 561	Integer, between 0 and 100 inclusive	50
“non_keywords option” on page 561	String	Empty string
“odbc_describe_binary_as_varbinary” on page 562	On, Off	Off
“odbc_distinguish_char_and_varchar option” on page 563	On, Off	Off
“oem_string option” on page 563	String (up to 128 bytes)	Empty string
“on_charset_conversion_failure option” on page 565	Ignore, Warning, Error	Ignore
“on_tsql_error option” on page 566	Stop, Conditional, Continue	Conditional Continue for jConnect connections
“optimization_goal option” on page 567	First-row or All-rows	All-rows
“optimization_level option” on page 568	0-15	9
“optimization_workload option” on page 569	Mixed, OLAP	Mixed

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Values	Default
“pinned_cursor_percent_of_cache option” on page 570	Integer, between 0-100	10
“post_login_procedure option” on page 571	String	Empty string
“precision option” on page 573	Integer, between 1 and 127 inclusive	30
“prefetch option” on page 573	Off, Conditional, Always	Conditional
“preserve_source_format option” on page 574	On, Off	On
“prevent_article_pkey_update option” on page 575	On, Off	On
“priority option” on page 575	Critical, High, Above Normal, Normal, Below normal, Low, Background	Normal
“progress_messages option” on page 576	Off, Formatted, Raw	Off
“query_mem_timeout option” on page 578	-1, 0, positive integer	-1
“quoted_identifier option” on page 579	On, Off	On
“read_past_deleted option” on page 580	On, Off	On
“recovery_time option” on page 580	Integer, in minutes	2
“remote_idle_timeout option” on page 581	Integer, in seconds	15
“request_timeout option” on page 582	Integer, 0 through 86400, in seconds	0
“reserved_keywords option” on page 583	String	Empty string
“return_date_time_as_string option” on page 584	On, Off	Off
“rollback_on_deadlock” on page 585	On, Off	On
“row_counts option” on page 585	On, Off	Off

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Values	Default
“scale option” on page 586	Integer, between 0 and 127, inclusive, and less than the value specified for the precision database option	6
“secure_feature_key” on page 587	String	NULL
“sort_collation option” on page 588	Internal, collation_name, or collation_id	Internal
“sql_flagger_error_level option” on page 589	Off, SQL:1992/Entry, SQL:1992/Intermediate, SQL:1992/Full, SQL:1999/Core, SQL:1999/Package, SQL:2003/Core, SQL:2003/Package, UltraLite	Off
“sql_flagger_warning_level option” on page 590	Off, SQL:1992/Entry, SQL:1992/Intermediate, SQL:1992/Full, SQL:1999/Core, SQL:1999/Package, SQL:2003/Core, SQL:2003/Package, UltraLite	Off
“string_rtruncation option” on page 598	On, Off	On
“subsume_row_locks option” on page 600	On, Off	On
“suppress_tds_debugging option” on page 600	On, Off	Off
“synchronize_mirror_on_commit option” on page 601	On, Off	Off
“tds_empty_string_is_null option” on page 601	On, Off	Off
“temp_space_limit_check option” on page 602	On, Off	On

Option	Values	Default
“time_format option” on page 603	String	HH:NN:SS.SSS
“time_zone_adjustment option” on page 604	Integer, or negative integer enclosed in quotation marks, or string, representing time in hours and minutes, preceded by + or -, enclosed in quotation marks	Set by either the client's or the database server's time zone, depending on the client's connection type
“timestamp_format option” on page 604	String	YYYY-MM-DD HH:NN:SS.SSS
“timestamp_with_time_zone_format option” on page 606	String	YYYY-MM-DD HH:NN:SS.SSS+HH:NN
“truncate_timestamp_values option” on page 608	On, Off	Off
“tsql_outer_joins option” on page 609	On, Off	Off
“tsql_variables option” on page 610	On, Off	Off On for Sybase Open Client and jConnect connections
“updatable_statement_isolation option” on page 610	0, 1, 2, 3	0
“update_statistics option” on page 611	On, Off	On
“user_estimates option” on page 612	Enabled, Disabled, Override-Magic	Override-Magic
“uuid_has_hyphens option” on page 613	On, Off	Off
“verify_password_function option” on page 614	String	Empty string
“wait_for_commit option” on page 617	On, Off	Off
“webservice_namespace_host option” on page 618	NULL, hostname-string	NULL

Compatibility options

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The following options allow you to make SQL Anywhere behavior compatible with Adaptive Server Enterprise, or to support both old behavior and allow ISO SQL/2008 behavior.

For further compatibility with Adaptive Server Enterprise, some of these options can be set for the duration of the current connection using the Transact-SQL SET statement instead of the SQL Anywhere SET OPTION statement. See "SET statement [T-SQL]" [*SQL Anywhere Server - SQL Reference*].

Default settings

The default setting for some of these options differs from the Adaptive Server Enterprise default setting. To ensure compatibility across your SQL Anywhere and Adaptive Server Enterprise databases, you should explicitly set each of the compatibility options listed in this section.

When a connection is made using the Sybase Open Client or JDBC interfaces, some option settings are explicitly set for the current connection to be compatible with Adaptive Server Enterprise. These options are listed in the following table.

Options for Sybase Open Client and JDBC connection compatibility with Adaptive Server Enterprise

Option	Setting
allow_nulls_by_default	Off
ansi_blanks	Off
ansi_substring	On
ansinull	On
chained	Off
continue_after_raiserror	On
escape_character	Off
on_tsql_error	Continue for jConnect connections
time_format	HH:NN:SS.SSS
timestamp_format	YYYY-MM-DD HH:NN:SS.SSS
tsql_outer_joins	Off
tsql_variables	On

Transact-SQL and SQL/2008 compatibility options

The following table lists the compatibility options, their allowed values, and their default settings.

Option	Values	Default
“allow_nulls_by_default option” on page 505	On, Off	On
“ansi_blanks option” on page 508	On, Off	Off
“ansi_close_cursors_on_rollback option” on page 509	On, Off	Off
“ansi_permissions option” on page 510	On, Off	On
“ansi_update_constraints option” on page 512	Off, Cursors, Strict	Cursors
“ansinull option” on page 513	On, Off	On
“chained option” on page 518	On, Off	On
“close_on_endtrans option” on page 520	On, Off	On
“continue_after_raiserror option” on page 524	On, Off	On
“conversion_error option” on page 525	On, Off	On
“date_format option” on page 528	String	YYYY-MM-DD
“date_order option” on page 530	MDY, YMD, DMY	YMD
“divide_by_zero_error option” on page 536	On, Off	On
“escape_character option” on page 536	Reserved	Reserved
“fire_triggers option” on page 538	On, Off	On
“isolation_level option” on page 544	0, 1, 2, 3	0
“nearest_century option” on page 561	Integer (between 0 and 100 inclusive)	50
“non_keywords option” on page 561	String (Comma-separated keywords list)	Empty string (No keywords turned off)
“on_tsql_error option” on page 566	Stop, Conditional, Continue	Conditional
“quoted_identifier option” on page 579	On, Off	On

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Values	Default
“reserved_keywords option” on page 583	String (Comma-separated keywords list)	Empty string (No keywords turned off)
“sql_flagger_error_level option” on page 589	Off, SQL:1992/Entry, SQL:1992/Intermediate, SQL:1992/Full, SQL:1999/Core, SQL:1999/Package, SQL:2003/Core, SQL:2003/Package, UltraLite	Off
“sql_flagger_warning_level option” on page 590	Off, SQL:1992/Entry, SQL:1992/Intermediate, SQL:1992/Full, SQL:1999/Core, SQL:1999/Package, SQL:2003/Core, SQL:2003/Package, UltraLite	Off
“string_rtruncation option” on page 598	On, Off	On
“time_format option” on page 603	String	HH:NN:SS.SSS
“timestamp_format option” on page 604	String	YYYY-MM-DD HH:NN:SS.SSS
“tsql_outer_joins option” on page 609	On, Off	Off
“tsql_variables option” on page 610	On, Off	Off

Synchronization options

The following database options can be set to configure SQL Anywhere databases used as MobiLink synchronization clients.

Option	Values	Default
“default_timestamp_increment option” on page 533	Integer (in microseconds from 1 to 1000000)	1

Option	Values	Default
“delete_old_logs option [SQL Remote]” on page 535	On, Off, Delay, *n* days	Off
“prevent_article_pkey_update option” on page 575	On, Off	On
“truncate_timestamp_values option” on page 608	On, Off	Off

SQL Remote options

The following options provide control over SQL Remote replication behavior.

Option	Values	Default
“blob_threshold option [SQL Remote]” on page 516	Integer (in bytes)	256
“compression option [SQL Remote]” on page 522	Integer, from -1 to 9	6
“delete_old_logs option [SQL Remote]” on page 535	On, Off, Delay, *n* days	Off
“external_remote_options [SQL Remote]” on page 537	On, Off	Off
“qualify_owners option [SQL Remote]” on page 578	On, Off	On
“quote_all_identifiers option [SQL Remote]” on page 579	On, Off	Off
“replication_error option [SQL Remote]” on page 581	Stored procedure name	(no procedure)
“replication_error_piece option [SQL Remote]” on page 582	Stored procedure name	(no procedure)
“save_remote_passwords option [SQL Remote]” on page 586	On, Off	On
“sr_date_format option [SQL Remote]” on page 591	*date-string*	yyyy/mm/dd

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Values	Default
“sr_time_format option [SQL Remote]” on page 592	*time-string*	hh:nn:ss.Ssssss
“sr_timestamp_format [SQL Remote]” on page 593	*timestamp-string*	yyyy/mm/dd hh:nn:ss.Ssssss
“sr_timestamp_with_time_zone_format [SQL Remote]” on page 593	*timestamp-with-time-zone-string*	yyyy/mm/dd hh:nn:ss.Ssssss +hh:nn
“subscribe_by_remote option [SQL Remote]” on page 599	On, Off	On
“verify_all_columns option [SQL Remote]” on page 613	On, Off	Off
“verify_threshold option [SQL Remote]” on page 617	Integer (in bytes)	1000

Alphabetical list of options

This section lists options alphabetically.

allow_nulls_by_default option

Controls whether new columns that are created without specifying either NULL or NOT NULL are allowed to contain NULL values.

Allowed values

On, Off

Default

On

Off for Sybase Open Client and jConnect connections

Remarks

The allow_nulls_by_default option is included for Transact-SQL compatibility. See “Setting options for Transact-SQL compatibility” [*SQL Anywhere Server - SQL Usage*].

See also

- "allow_nulls_by_default connection property" on page 619
- "Characteristics of Sybase Open Client and jConnect connections" on page 1173
- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "ALTER TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "Setting options for Transact-SQL compatibility" [*SQL Anywhere Server - SQL Usage*]
- "Creating compatible tables" [*SQL Anywhere Server - SQL Usage*]
- "Compatibility options" on page 500
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]

allow_read_client_file option

Controls whether to allow the reading of files on a client computer.

Allowed values

On, Off

Default

Off

Scope

DBA authority required.

Remarks

This option must be enabled to read from files on a client computer, for example using the READ_CLIENT_FILE function.

See also

- "Accessing data on client computers" [*SQL Anywhere Server - SQL Usage*]
- "READ_CLIENT_FILE function [String]" [*SQL Anywhere Server - SQL Reference*]
- "READCLIENTFILE authority" on page 446
- "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "isql_allow_read_client_file option [Interactive SQL]" on page 745
- "allow_write_client_file option" on page 508
- "isql_allow_write_client_file option [Interactive SQL]" on page 746
- "Client-side data security" [*SQL Anywhere Server - SQL Usage*]
- "allow_read_client_file connection property" on page 619

allow_snapshot_isolation option

Controls whether snapshot isolation is enabled or disabled.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Allowed values

On, Off

Default

Off

Scope

Can be set for the PUBLIC group only. DBA authority required.

Remarks

This option controls whether snapshot isolation is enabled for the database. Once this option is set to On, the database server starts recording the original versions of updated rows in the temporary file if a transaction uses snapshot isolation.

If there are transactions in progress when the setting of the allow_snapshot_isolation option is changed, then the change does not take effect immediately. Any transactions that are running when the option setting is changed from Off to On must complete before snapshots can be used. When the setting of the option is changed from On to Off, any outstanding snapshots are allowed to complete before the database server stops collecting version information, and new snapshots are not initiated.

You can view the current snapshot isolation setting for a database by querying the value of the SnapshotIsolationState database property:

```
SELECT DB_PROPERTY ( 'SnapshotIsolationState' );
```

The SnapshotIsolationState property has one of the following values:

- **On** Snapshot isolation is enabled for the database.
- **Off** Snapshot isolation is disabled for the database.
- **in_transition_to_on** Snapshot isolation will be enabled once the current transactions complete.
- **in_transition_to_off** Snapshot isolation will be disabled once the current transactions complete.

See also

- “isolation_level option” on page 544
- “updatable_statement_isolation option” on page 610
- “Snapshot isolation” [*SQL Anywhere Server - SQL Usage*]
- “Isolation levels and consistency” [*SQL Anywhere Server - SQL Usage*]
- “Enabling snapshot isolation” [*SQL Anywhere Server - SQL Usage*]
- “allow_snapshot_isolation connection property” on page 619

Example

The following statement enables snapshot isolation for a database:

```
SET OPTION PUBLIC.allow_snapshot_isolation = 'On';
```

allow_write_client_file option

Controls whether to allow the writing of files to a client computer.

Allowed values

On, Off

Default

Off

Scope

DBA authority required.

Remarks

This option must be enabled to write files to a client computer, for example using the WRITE_CLIENT_FILE function.

See also

- "Accessing data on client computers" [*SQL Anywhere Server - SQL Usage*]
- "WRITE_CLIENT_FILE function [String]" [*SQL Anywhere Server - SQL Reference*]
- "WRITECLIENTFILE authority" on page 447
- "UNLOAD statement" [*SQL Anywhere Server - SQL Reference*]
- "isql_allow_write_client_file option [Interactive SQL]" on page 746
- "allow_read_client_file option" on page 506
- "isql_allow_read_client_file option [Interactive SQL]" on page 745
- "Client-side data security" [*SQL Anywhere Server - SQL Usage*]
- "allow_write_client_file connection property" on page 620

ansi_blanks option

Controls behavior when character data is truncated at the client side.

Allowed values

On, Off

Default

Off

Remarks

The ansi_blanks option has no effect unless the database ignores trailing blanks in string comparisons and pads strings that are fetched into character arrays. It forces a truncation error whenever a value of data type CHAR(*N*) is read into a C char[*M*] variable for values of *N* greater than or equal to *M*. With ansi_blanks set to Off, a truncation error occurs only when at least one non-blank character is truncated.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For embedded SQL with the ansi_blanks option set to On, when you supply a value of data type DT_STRING, you must set the sqllen field to the length of the buffer containing the value (at least the length of the value plus space for the terminating null character). With ansi_blanks set to Off, the length is determined solely by the position of the NULL character. The value of the ansi_blanks option is determined when the connection is established. Changing the option once the connection has been made does not affect this sqllen embedded SQL behavior.

When a database is blank padded, this option controls truncation warnings sent to the client if the expression being fetched is CHAR or NCHAR (not VARCHAR or NVARCHAR) and it is being fetched into a char or nchar (not VARCHAR or NVARCHAR) host variable. If these conditions hold and the host variable is too small to hold the fetched expression once it is blank padded to the expression's maximum length, a truncation warning is raised and the indicator contains the minimum number of bytes required to hold the fetched expression if it is blank padded to its maximum length. If the expression is CHAR(N) or NCHAR(N), the indicator may be set to a value other than N to take in account character set translation of the value returned and character length semantics.

See also

- "Compatibility options" on page 500
- "ansi_blanks connection property" on page 620
- "Characteristics of Sybase Open Client and jConnect connections" on page 1173
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]

ansi_close_cursors_on_rollback option

Controls whether cursors that were opened WITH HOLD are closed when a ROLLBACK is performed.

Allowed values

On, Off

Default

Off

Remarks

The draft SQL/3 standard requires all cursors be closed when a transaction is rolled back. By default, on a rollback SQL Anywhere closes only those cursors that were opened without a WITH HOLD clause. This option allows you to force closure of all cursors.

The close_on_endtrans option overrides the ansi_close_cursors_on_rollback option.

See also

- "close_on_endtrans option" on page 520
- "Compatibility options" on page 500
- "ansi_close_cursors_on_rollback connection property" on page 620
- "Cursors and transactions" [*SQL Anywhere Server - Programming*]

ansi_permissions option

Controls permissions checking for DELETE and UPDATE statements.

Allowed values

On, Off

Default

On

Scope

Can be set for the PUBLIC group only. Takes effect immediately. DBA authority required.

Remarks

With ansi_permissions set to On, the SQL/2008 permissions requirements for DELETE and UPDATE statements are checked. The default value is Off in Adaptive Server Enterprise. The following table outlines the differences.

SQL statement	Permissions required with ansi_permissions off	Permissions required with ansi_permissions on
UPDATE	UPDATE permission on the columns where values are being set	UPDATE permission on the columns where values are being set SELECT permission on all columns appearing in the WHERE clause SELECT permission on all columns on the right side of the SET clause
DELETE	DELETE permission on the table	DELETE permission on the table SELECT permission on all columns appearing in the WHERE clause

The ansi_permissions option can be set only for the PUBLIC group. No private settings are allowed.

See also

- “Compatibility options” on page 500
- “ansi_permissions connection property” on page 620
- “SET statement [T-SQL]” [*SQL Anywhere Server - SQL Reference*]

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

ansi_substring option

Controls the behavior of the SUBSTRING (SUBSTR) function when negative values are provided for the start or length parameters.

Allowed values

On, Off

Default

On

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

When the ansi_substring option is set to On, the behavior of the SUBSTRING function corresponds to ANSI/ISO SQL/2008 behavior. A negative or zero start offset is treated as if the string were padded on the left with non-characters, and gives an error if a negative length is provided.

When this option is set to Off, the behavior of the SUBSTRING function is the same as in previous releases of SQL Anywhere: a negative start offset means an offset from the end of the string, and a negative length means the desired substring ends length characters to the left of the starting offset. Also, using a start offset of 0 is equivalent to a start offset of 1.

The setting of this option does not affect the behavior of the BYTE_SUBSTR function. It is recommended that you avoid using non-positive start offsets or negative lengths with the SUBSTRING function. Where possible, use the LEFT or RIGHT functions instead.

See also

- "SUBSTRING function [String]" [*SQL Anywhere Server - SQL Reference*]
- "LEFT function [String]" [*SQL Anywhere Server - SQL Reference*]
- "RIGHT function [String]" [*SQL Anywhere Server - SQL Reference*]
- "Compatibility options" on page 500
- "ansi_substring connection property" on page 620

Example

The following examples show the difference in the values returned by the SUBSTRING function based on the setting of the ansi_substring option.

```
SUBSTRING( 'abcdefgh',-2,4 );
  ansi_substring = Off ==> 'gh' // substring starts at second-last character
  ansi_substring = On  ==> 'a'  // takes the first 4 characters of
                                // ???abcdefgh and discards all ?
SUBSTRING( 'abcdefgh',4,-2 );
  ansi_substring = Off ==> 'cd'
  ansi_substring = On  ==> value -2 out of range for destination

SUBSTRING( 'abcdefgh',0,4 );
  ansi_substring = Off ==> 'abcd'
  ansi_substring = On  ==> 'abc'
```

ansi_update_constraints option

Controls the range of updates that are permitted.

Allowed values

Off, Cursors, Strict

Default

Cursors

Remarks

SQL Anywhere provides several extensions that allow updates that are not permitted by the ANSI SQL standard. These extensions provide powerful, efficient mechanisms for performing updates. However, they may cause unexpected behavior. This behavior can produce anomalies such as lost updates if the user application is not designed to expect the behavior of these extensions.

The ansi_update_constraints option controls whether updates are restricted to those permitted by the SQL/2008 standard.

If the option is set to Strict, the following updates are prevented:

- Updates of cursors containing JOINS
- Updates of columns that appear in an ORDER BY clause
- The FROM clause is not allowed in UPDATE statements

If the option is set to Cursors, these same restrictions are in place, but only for cursors. If a cursor is not opened with FOR UPDATE or FOR READ ONLY, the database server chooses updatability based on the SQL/2008 standard. If the ansi_update_constraints option is set to Cursors or Strict, cursors containing an ORDER BY clause default to FOR READ ONLY; otherwise, they continue to default to FOR UPDATE.

For ODBC, JDBC, ADO.NET, and OLE DB, statement updatability is explicitly READ ONLY.

For embedded SQL, statement updatability is explicitly READ ONLY unless the SQL preprocessor -m HISTORICAL option is specified.

By default, stored procedure cursors are not explicitly OR UPDATE or READ ONLY.

See also

- "UPDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "SQL preprocessor" [*SQL Anywhere Server - Programming*]
- "Compatibility options" on page 500
- "ansi_update_constraints connection property" on page 620
- "Updating and deleting rows through a cursor" [*SQL Anywhere Server - Programming*]

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

ansinull option

Controls the interpretation of NULL values.

Allowed values

On, Off

Default

On

Remarks

This option is implemented primarily for Transact-SQL (Adaptive Server Enterprise) compatibility. The ansinull option affects the results of comparison predicates with NULL constants, and also affects warnings issued for grouped queries over NULL values.

With ansinull set to On, ANSI three-valued logic is used for all comparison predicates in a WHERE or HAVING clause, or in an On condition. Any comparisons with NULL using **=** or **!=** evaluate to unknown.

Setting ansinull to Off means that SQL Anywhere uses two-valued logic for the following four conditions:

expr = NULL

expr != NULL

expr = @var // @var is a procedure variable, or a host variable

expr != @var

In each case, the predicate evaluates to either true or false—never unknown. In such comparisons, the NULL value is treated as a special value in each domain, and an equality (=) comparison of two NULL values yields true. Note that the expression *expr* must be a relatively simple expression, referencing only columns, variables, and literals; subqueries and functions are not permitted.

With ansinull set to On, the evaluation of any aggregate function, except COUNT(*), on an expression that contains at least one NULL value, may generate the warning `null value eliminated in aggregate function` (SQLSTATE=01003). With ansinull set to Off, this warning does not appear.

Limitations

- Setting ansinull to Off affects only WHERE, HAVING, or ON predicates in SELECT, UPDATE, DELETE, and INSERT statements. The semantics of comparisons in a CASE or IF statement, or in IF expressions, are unaffected.
- Adaptive Server Enterprise 12.5 introduced a change in the behavior of LIKE predicates with a NULL pattern string when ansinull is set to Off. In SQL Anywhere, LIKE predicates remain unaffected by the setting of ansinull.

See also

- "Compatibility options" on page 500
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]
- "SET statement [T-SQL]" [*SQL Anywhere Server - SQL Reference*]
- "ansinull connection property" on page 620
- "Characteristics of Sybase Open Client and jConnect connections" on page 1173

auditing option

Enables and disables auditing in the database.

Allowed values

On, Off

Default

Off

Scope

Can be set for the PUBLIC group only. Takes effect immediately. DBA authority required.

Remarks

This option turns auditing on and off.

Auditing is the recording of details about many events in the database in the transaction log. Auditing provides some security features, at the cost of some performance. When you turn on auditing for a database, you cannot stop using the transaction log. You must turn auditing off before you turn off the transaction log. Databases with auditing on cannot be started in read-only mode.

For the auditing option to work, you must set the auditing option to On, and also specify which types of information you want to audit using the sa_enable_auditing_type system procedure. Auditing will not take place if either of the following are true:

- The auditing option is set to Off
- Auditing options have been disabled

If you set the auditing option to On, and do not specify auditing options, all types of auditing information are recorded. Alternatively, you can choose to record any combination of the following: permission checks, connection attempts, DDL statements, public options, and triggers using the sa_enable_auditing_type system procedure.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Auditing database activity" on page 1123
- "sa_audit_string system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_enable_auditing_type system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_disable_auditing_type system procedure" [*SQL Anywhere Server - SQL Reference*]
- "auditing connection property" on page 621

Example

The following statement turns on auditing for the database.

```
SET OPTION PUBLIC.auditing = 'On';
```

auditing_options option

This option is reserved for system use. Do not change the setting of this option.

background_priority option [deprecated]

Deprecated. Limits impact on the performance of connections other than the current connection.

Allowed values

On, Off

Default

Off

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

If you set this option temporarily, that setting applies to the current connection only. Different connections under the same user ID can have different settings for this option.

Intra-query parallelism is not used for connections with background_priority set to on. See "Parallelism during query execution" [*SQL Anywhere Server - SQL Usage*].

Remarks

Setting this option to On causes requests to execute at the Background priority level. When this option is set to Off, requests execute at the value specified by the Priority option.

See also

- "priority option" on page 575
- "max_priority option" on page 555
- "background_priority connection property" on page 621

blob_threshold option [SQL Remote]

Controls the size of value that the Message Agent treats as a long object (BLOB).

Allowed values

Integer, in bytes

Default

256

Remarks

Any value longer than the blob_threshold option is replicated as a BLOB. That is, it is broken into pieces and replicated in chunks before being reconstituted using a SQL variable and concatenating the pieces at the recipient site.

Each SQL statement must fit within a message, so you should not set the value of this option to a size larger than your message size (50 KB by default).

See also

- "SQL Remote options" [*SQL Remote*]
- "BLOBs" [*SQL Remote*]

blocking option

Controls the behavior in response to locking conflicts.

Allowed values

On, Off

Default

On

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

If the blocking option is set to On, any transaction attempting to obtain a lock that conflicts with an existing lock held by another transaction waits until every conflicting lock is released or until the blocking_timeout is reached. If the lock is not released within blocking_timeout milliseconds, then an error is returned for the waiting transaction. If the blocking option is set to Off, the transaction that attempts to obtain a conflicting lock receives an error.

The value for the blocking connection property can be temporarily set to Off during certain operations (for example, refreshing a materialized view), which can then also cause a SQLSTATE 42W18 and SQLCODE -210. See "User '%1' has the row in '%2' locked" [*Error Messages*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "blocking_timeout option" on page 517
- "The blocking option" [*SQL Anywhere Server - SQL Usage*]
- "blocking connection property" on page 621

blocking_others_timeout option

Specifies the amount of time that another connection can block on the current connection's row and table locks before the current connection is rolled back. This option can be used to prevent a low priority task from blocking other connections for longer than the specified time.

Allowed values

Integer, in milliseconds

Default

0

Scope

Only supported by command sequence connections (not supported by TDS or HTTP connections or by events). Can be set as a temporary option only, for the duration of the current connection.

Remarks

When this option is set to 0, other connections can block on the current connection for an indefinite period of time.

If another connection is blocked on the current connection for longer than the number of milliseconds specified by this option, the current connection is rolled back. If the current connection is in the middle of a request, the request is interrupted. An error is returned if the connection was rolled back. The error may be returned twice if a request needs to be interrupted. If the connection was idle, the connection is rolled back immediately. The application must be prepared to fail or retry after a delay if it is rolled back.

Note
Locks created by LOCK TABLE *table-name* WITH HOLD are rolled back. Once a connection receives an error indicating that an operation was rolled back because of a blocking_others_timeout error, that connection receives the error on every request until the connection disconnects.

See also

- "blocking_others_timeout connection property" on page 621
- "blocking_timeout option" on page 517
- "Transaction blocking and deadlock" [*SQL Anywhere Server - SQL Usage*]
- "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]

blocking_timeout option

Controls how long a transaction waits to obtain a lock.

Allowed values

Integer, in milliseconds

Default

0

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

When the blocking option is set to On, any transaction attempting to obtain a lock that conflicts with an existing lock waits for blocking_timeout milliseconds for the conflicting lock to be released. If the lock is not released within blocking_timeout milliseconds, then an error is returned for the waiting transaction.

Setting this option to 0 forces all transactions attempting to obtain a lock to wait until all conflicting transactions release their locks.

See also

- "blocking option" on page 516
- "blocking_others_timeout connection property" on page 621
- "request_timeout option" on page 582
- "Deadlock" [*SQL Anywhere Server - SQL Usage*]
- "blocking_timeout connection property" on page 621

chained option

Controls the transaction mode in the absence of a BEGIN TRANSACTION statement.

Allowed values

On, Off

Default

On

Off for Sybase Open Client and jConnect connections

Remarks

Controls the Transact-SQL transaction mode. In Unchained mode (chained=Off), each statement is committed individually unless an explicit BEGIN TRANSACTION statement is executed to start a transaction. In chained mode (chained=On) a transaction is implicitly started before any data retrieval or modification statement.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Compatibility options" on page 500
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Autocommit implementation details" [*SQL Anywhere Server - Programming*]
- "Using transactions" [*SQL Anywhere Server - SQL Usage*]
- "chained connection property" on page 622

checkpoint_time option

Sets the maximum number of minutes that the database server will run without doing a checkpoint.

Allowed values

Integer

Default

60

Scope

Can be set for the PUBLIC group only. DBA authority required. You must shut down and restart the database server for the change to take effect.

Remarks

This option is used with the recovery_time option to decide when checkpoints should be done.

See also

- "Understanding the checkpoint log" on page 25
- "recovery_time option" on page 580
- "How the database server decides when to checkpoint" on page 924
- "-gc dbeng12/dbsrv12 server option" on page 184
- "-m dbeng12/dbsrv12 server option" on page 205
- "CHECKPOINT statement" [*SQL Anywhere Server - SQL Reference*]
- "checkpoint_time connection property" on page 622

cis_option option

Controls whether debugging information for remote data access appears in the database server messages window.

Allowed values

0, 7

Default

0

Scope

Can be set for an individual connection or for the PUBLIC group.

Remarks

This option controls whether information about how queries are executed on a remote database appears in the database server messages window when using remote data access. Set this option to 7 to see debugging information in the database server messages window. When this option is set to 0 (the default), debugging information for remote data access does not appear in the database server messages window.

Once you have turned on remote tracing, the tracing information appears in the database server messages window. You can log this output to a file by specifying the -o server option when you start the database server. See "-o dbeng12/dbsrv12 server option" on page 208.

See also

- "General problems with queries" [*SQL Anywhere Server - SQL Usage*]
- "Connectivity tests" [*SQL Anywhere Server - SQL Usage*]
- "cis_option connection property" on page 622

cis_rowset_size option

Sets the number of rows that are returned from remote servers for each fetch.

Allowed values

Integer

Default

50

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect when a new connection is made to a remote server.

Remarks

This option sets the ODBC FetchArraySize value when using ODBC to connect to a remote database server.

See also

- "cis_rowset_size connection property" on page 622

close_on_endtrans option

Controls the closing of cursors at the end of a transaction.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Allowed values

On, Off

Default

On

Off for jConnect connections

Remarks

When close_on_endtrans is set to On, cursors are closed whenever a transaction is committed unless the cursor was opened WITH HOLD. The behavior when a transaction is rolled back is governed by the ansi_close_cursors_on_rollback option.

When close_on_endtrans is set to Off, cursors are not closed at either a commit or a rollback, regardless of the ansi_close_cursors_on_rollback option setting or whether the cursor was opened WITH HOLD or not.

Setting this to Off provides Adaptive Server Enterprise compatible behavior.

See also

- "ansi_close_cursors_on_rollback option" on page 509
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Compatibility options" on page 500
- "SET statement [T-SQL]" [*SQL Anywhere Server - SQL Reference*]
- "Characteristics of Sybase Open Client and jConnect connections" on page 1173
- "close_on_endtrans connection property" on page 623

collect_statistics_on_dml_updates option

Controls the gathering of statistics during the execution of data-altering DML statements such as INSERT, DELETE, and UPDATE.

Allowed values

On, Off

Default

On

Remarks

The database server updates statistics during normal statement execution and uses the gathered statistics to self-tune column statistics. Set the collect_statistics_on_dml_updates option to Off to disable the updating of statistics during the execution of data-altering DML statements such as INSERT, DELETE, and UPDATE.

Under normal circumstances, it should not be necessary to turn this option off. However, in environments where significantly large amounts of data are frequently changing, setting this option to Off may improve performance—assuming update_statistics is also set to On.

The difference between the collect_statistics_on_dml_updates option and the update_statistics option is that the update_statistics option compares the actual number of rows that satisfy a predicate with the number of rows that are estimated to satisfy the predicate, and then updates the estimates. The collect_statistics_on_dml_updates option modifies the column statistics based on the values of the specific rows that are inserted, updated, or deleted.

See also

- "update_statistics option" on page 611
- "Updating column statistics to improve optimizer performance" [*SQL Anywhere Server - SQL Usage*]
- "collect_statistics_on_dml_updates connection property" on page 623

compression option [SQL Remote]

Sets the level of compression for SQL Remote messages.

Allowed values

Integer, from -1 to 9

Default

6

Remarks

The values have the following meanings:

- **-1** Send messages in version 5 format. The Message Agent from version 5 cannot read messages sent by the Message Agent from version 6 and later. You should ensure that the compression option is set to -1 until all Message Agents in your system are upgraded to version 6 or later.
- **0** No compression.
- **1 to 9** Increasing degrees of compression. Creating messages with high compression can take longer than creating messages with low compression.

See also

- "SQL Remote options" [*SQL Remote*]
- "Controlling message size" [*SQL Remote*]

conn_auditing option

Controls whether auditing is enabled or disabled for each connection when the auditing option is set to On.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Allowed values

On, Off

Default

On

Scope

Can be set as a temporary option only, for the duration of the current connection. DBA authority required.

Remarks

The setting of the conn_auditing option is only respected when it is set in a login procedure (specified by the login_procedure database option). Setting conn_auditing to On turns on auditing for the connection. However, auditing information is not recorded unless the auditing option is also set to On. You can execute the following statement to determine whether a connection is being audited:

```
SELECT CONNECTION_PROPERTY ( 'conn_auditing' );
```

See also

- "Controlling auditing" on page 1123
- "auditing option" on page 514
- "login_procedure option" on page 549
- "conn_auditing connection property" on page 624

connection_authentication option

Specifies an authentication string that is used to verify the application signature against the database signature for authenticated applications.

Allowed values

String

Default

Empty string

Scope

Can be set for an individual connection only.

Remarks

This option only takes effect when you are using the OEM Edition of the SQL Anywhere database server.

Authenticated applications must set the connection_authentication database option for every connection immediately after the connection is established. If the signature is verified, the connection is authenticated and has no restrictions on its activities beyond those imposed by the SQL permissions. If the signature is not verified, the connection is limited to those actions permitted by unauthenticated applications.

The connection_authentication option must be set for the duration of the current connection only by using the TEMPORARY keyword. The following SQL statement authenticates the connection:

```
SET TEMPORARY OPTION connection_authentication =
     'company = company-name;
     application = application-name;
     signature = application-signature';
```

The *company-name* and *application-name* must match those in the database authentication statement. The *application-signature* is the application signature that you obtained from Sybase.

If your company name has quotation marks, apostrophes, or other special characters, you must double them in the string for it to be accepted.

For more information about configuring and using the OEM Edition of SQL Anywhere, see "Running authenticated SQL Anywhere applications" on page 69.

See also

- "database_authentication" on page 527
- "connection_authentication connection property" on page 624

Example

The following example specifies an authentication string that contains special characters:

```
SET TEMPORARY OPTION connection_authentication=
   'Company = Joe''s Garage;
    Application = Joe''s Program;
    Signature = 0fa55157edb8e14d818e...';
```

continue_after_raiserror option

Controls behavior following a RAISERROR statement.

Allowed values

On, Off

Default

On

Remarks

The RAISERROR statement is used within procedures and triggers to generate an error. When this option is set to Off, the execution of the procedure or trigger is stopped whenever the RAISERROR statement is encountered.

If you set the continue_after_raiserror option to On, the RAISERROR statement no longer signals an execution-ending error. Instead, the RAISERROR status code and message are stored and the most recent RAISERROR is returned when the procedure completes. If the procedure that caused the RAISERROR was called from another procedure, the RAISERROR is not returned until the outermost calling procedure ends.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Intermediate RAISERROR statuses and codes are lost after the procedure ends. If, at return time, an error occurs along with the RAISERROR, then the information for the new error is returned and the RAISERROR information is lost. The application can query intermediate RAISERROR statuses by examining the @@error global variable at different execution points.

The setting of the continue_after_raiserror option is used to control behavior following a RAISERROR statement only if the on_tsql_error option is set to Conditional (the default). If you set the on_tsql_error option to Stop or Continue, the on_tsql_error setting takes precedence over the continue_after_raiserror setting.

See also

- "on_tsql_error option" on page 566
- "Compatibility options" on page 500
- "RAISERROR statement" [*SQL Anywhere Server - SQL Reference*]
- "continue_after_raiserror connection property" on page 624

conversion_error option

Controls the reporting of data type conversion failures on fetching information from the database.

Allowed values

On, Off

Default

On

Remarks

This option controls whether data type conversion failures, when data is fetched from the database or inserted into the database, are reported by the database as errors (conversion_error set to On) or as a warning (conversion_error set to Off).

When conversion_error is set to On, the SQLE_CONVERSION_ERROR error is generated. If the option is set to Off, the warning SQLE_CANNOT_CONVERT is produced.

If conversion errors are reported as warnings only, the NULL value is used in place of the value that could not be converted. In embedded SQL, an indicator variable is set to -2 for the column or columns that cause the error.

See also

- "Using indicator values for conversion errors" [*SQL Anywhere Server - Programming*]
- "Handling conversion errors during import" [*SQL Anywhere Server - SQL Usage*]
- "Compatibility options" on page 500
- "conversion_error connection property" on page 624

cooperative_commit_timeout option

Governs when a COMMIT entry in the transaction log is written to disk.

Allowed values

Integer, in milliseconds

Default

250

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

This option has meaning only when cooperative_commits is set to On. The database server waits for the specified number of milliseconds for other connections to fill a page of the log before writing to disk. The default setting is 250 milliseconds.

See also

- "cooperative_commits option" on page 526
- "delayed_commit_timeout option" on page 534
- "COMMIT statement" [*SQL Anywhere Server - SQL Reference*]
- "cooperative_commit_timeout connection property" on page 624

cooperative_commits option

Controls when commits are written to disk.

Allowed values

On, Off

Default

On

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

If cooperative_commits is set to Off, a COMMIT is written to disk when the database server receives it, and the application is then allowed to continue.

If cooperative_commits is set to On (the default) and if there are other active connections, the database server does not immediately write the COMMIT to the disk. Instead, the application waits for up to the

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

maximum length set by the cooperative_commit_timeout option for something else to put on the pages before they are written to disk.

Setting cooperative_commits to On, and increasing the cooperative_commit_timeout setting, increases overall database server throughput by cutting down the number of disk I/Os, but at the expense of a longer turnaround time for each individual connection.

If both cooperative_commits and delayed_commits are set to On, and the cooperative_commit_timeout interval passes without the pages getting written, the application is resumed (as if the commit had worked), and the remaining interval (delayed_commit_timeout - cooperative_commit_timeout) is used as a delayed_commits interval. The pages are then written, even if they are not full.

See also

- "delayed_commits option" on page 534
- "COMMIT statement" [*SQL Anywhere Server - SQL Reference*]
- "cooperative_commits connection property" on page 624

database_authentication

Sets the authentication string for a database.

Allowed values

String

Default

Empty string

Scope

Can be set for the PUBLIC group only. You must restart the database for this option to take effect.

Remarks

This option only takes effect when you are using the OEM Edition of the SQL Anywhere database server.

When a database is authenticated, only connections that specify the correct authentication signature can perform operations on the database. Connections that are not authenticated operate in read-only mode. You must use the OEM Edition of SQL Anywhere if you want to use authenticated databases.

To authenticate a database, set the database_authentication option for the database:

```
SET OPTION PUBLIC.database_authentication =
 'company = company-name;
 application = application-name;
 signature = database-signature';
```

The *company-name* and *application-name* arguments are the values you supplied to Sybase when obtaining your signature, and *database-signature* is the database signature that you received from Sybase.

If your company name has quotation marks, apostrophes, or other special characters, you must double them in the string for it to be accepted.

When the database server loads an authenticated database, it displays a message in the database server messages window describing the authenticated company and application. You can check that this message is present to verify that the database_authentication option has taken effect. The message has the following form:

```
This database is licensed for use with:
Application: application-name
Company: company-name
```

You can store the authentication statement in a SQL script file to avoid having to type in the long signature repeatedly. If you store the authentication statement in the file *install-dir\scripts\authenticate.sql*, it is applied whenever you create, rebuild, or upgrade a database.

For more information about configuring and using the OEM Edition of SQL Anywhere, see "Running authenticated SQL Anywhere applications" on page 69.

See also

- "connection_authentication option" on page 523
- "database_authentication connection property" on page 625

Example

```
SET OPTION PUBLIC.database_authentication =
     'company = MyCompany;
     application = MySQLAnywhereApp;
     signature = 0fa55157edb8e14d818e';
```

date_format option

Sets the format for dates retrieved from the database.

Allowed values

String

Default

YYYY-MM-DD

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

The format is a string using the following symbols:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Symbol	Description
YY	Two digit year
YYYY	Four digit year
MM	Two digit month
MMM[m...]	Character short form for months
D	Single digit day of week, (1 = Sunday, 7 = Saturday)
DD	Two digit day of month
DDD[d...]	Character short form for days of the week
JJJ	Day of the year, from 1 to 366

Each symbol is substituted with the appropriate data for the date that is being formatted.

For more information about controlling the interpretation of date formats, see "date_order option" on page 530.

If the character data is multibyte, the length of each symbol reflects the number of characters, not the number of bytes. For example, the 'mmm' symbol specifies a length of three characters for the month.

For symbols that represent character data (such as *mmm*), you can control the case of the output as follows:

- Type the symbol in all uppercase to have the format appear in all uppercase. For example, MMM produces JAN.
- Type the symbol in all lowercase to have the format appear in all lowercase. For example, mmm produces jan.
- Type the symbol in mixed case to have SQL Anywhere choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.

For symbols that represent numeric data, you can control zero-padding with the case of the symbols:

- Type the symbol in same-case (such as MM or mm) to allow zero padding. For example, yyyy/mm/dd could produce 2002/01/01.
- Type the symbol in mixed case (such as Mm) to suppress zero padding. For example, yyyy/Mm/Dd could produce 2002/1/1.

Note
If you change the setting for date_format in a way that re-orders the date format, be sure to change the date_order option to reflect the same change, and vice versa. See "date_order option" on page 530.

See also

- "Comparing dates in search conditions" [*SQL Anywhere Server - SQL Usage*]
- "time_format option" on page 603
- "timestamp_format option" on page 604
- "timestamp_with_time_zone_format option" on page 606
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]
- "date_format connection property" on page 625

Example

The following table illustrates date_format settings, together with the output from the following statement, executed on Monday, April 14, 2008.

```
SELECT CAST( CURRENT DATE AS VARCHAR );
```

date_format	CURRENT DATE
yyyy/mm/dd/ddd	2008/04/14/mon
yyyy/Mm/Dd/ddd	2008/4/14/mon
jjj	105
mmm yyyy	apr 2008
Mmm yyyy	Apr 2008
mm-yyyy	04-2008

date_order option

Controls the interpretation of date formats.

Allowed values

MDY, YMD, DMY

Default

YMD (this corresponds to ISO date format specifications)

For Sybase Open Client and jConnect connections, the default is set to MDY

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

The database option date_order is used to determine whether 10/11/12 is Oct 11 1912, Nov 12 1910, or Nov 10 1912.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Note
If you change the setting for date_order in a way that re-orders the date format, be sure to change the date_format and timestamp_format options to reflect the same change, and vice versa. See "date_format option" on page 528, and "timestamp_format option" on page 604.

See also

- "Comparing dates in search conditions" [*SQL Anywhere Server - SQL Usage*]
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]
- "date_order connection property" on page 625

debug_messages option

Controls whether MESSAGE statements that include a DEBUG ONLY clause are executed.

Allowed values

On, Off

Default

Off

Remarks

This option allows you to control the behavior of debugging messages in stored procedures and triggers that contain a MESSAGE statement with the DEBUG ONLY clause specified. By default, this option is set to Off and debugging messages do not appear when the MESSAGE statement is executed. By setting debug_messages to On, you can enable the debugging messages in all stored procedures and triggers.

Note
DEBUG ONLY messages are inexpensive when the debug_messages option is set to Off, so these statements can usually be left in stored procedures on a production system. However, they should be used sparingly in locations where they would be executed frequently; otherwise, they may result in a small performance penalty.

See also

- "MESSAGE statement" [*SQL Anywhere Server - SQL Reference*]
- "debug_messages connection property" on page 625

dedicated_task option

Dedicates a request handling task to handling requests from a single connection.

Allowed values

On, Off

Default

Off

Scope

Can be set as a temporary option only, for the duration of the current connection. DBA authority required.

Remarks

When the dedicated_task connection option is set to On, a request handling task is dedicated exclusively to handling requests for the connection. By pre-establishing a connection with this option enabled, you will be able to gather information about the state of the database server if it becomes otherwise unresponsive.

See also

- "dedicated_task connection property" on page 625

default_dbspace option

Changes the default dbspace in which tables are created.

Allowed values

String

Default

Empty string

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

For each database, you can create up to twelve dbspaces in addition to the system (main) dbspace. When a table is created without specifying a dbspace, the dbspace named by this option setting is used. If this option is not set, is set to the empty string, or is set to system, then the system dbspace is used.

When you create temporary tables or indexes, they are always placed in the TEMPORARY dbspace, regardless of the setting of the default_dbspace option. If you specify the IN clause when creating a base table, the dbspace specified by the IN clause is used, rather than the dbspace specified by the default_dbspace option.

If all tables are created in a location other than the system dbspace, then the system dbspace is only used for the checkpoint log and system tables. This is useful if you want to put the checkpoint log on a separate disk from the rest of your database objects for performance reasons. You can place the checkpoint log in a separate disk by changing all CREATE TABLE statements to specify the dbspace, or by changing this option before creating any tables.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Using additional dbspaces" on page 16
- "Place different files on different devices" [*SQL Anywhere Server - SQL Usage*]
- "CREATE DBSPACE statement" [*SQL Anywhere Server - SQL Reference*]
- "ALTER DBSPACE statement" [*SQL Anywhere Server - SQL Reference*]
- "Creating dbspaces" on page 18
- "-ds dbeng12/dbsrv12 database option" on page 254
- "default_dbspace connection property" on page 625

Example

In the following example, a new dbspace named MyLibrary is created. The default dbspace is then set to the MyLibrary dbspace and the table LibraryBooks is stored in the MyLibrary dbspace instead of the system dbspace.

```
CREATE DBSPACE MyLibrary
AS 'c:\\dbfiles\\library.db';
SET OPTION default_dbspace = 'MyLibrary';
CREATE TABLE LibraryBooks (
  title CHAR(100),
  author CHAR(50),
  isbn CHAR(30),
);
```

default_timestamp_increment option

Specifies the number of microseconds to add to a column of type TIMESTAMP to keep values in the column unique.

Allowed values

Integer, between 1 and 1000000 inclusive

Default

1

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

Since a TIMESTAMP value is precise to six decimal places in SQL Anywhere, by default 1 microsecond (0.000001 of a second) is added to differentiate between two identical TIMESTAMP values.

Some software, such as Microsoft Access, truncates TIMESTAMP values to three decimal places, making valid comparisons a problem. You can set the truncate_timestamp_values option to On to specify the number of decimal place values SQL Anywhere stores to maintain compatibility.

For MobiLink synchronization, if you are going to set this option, it must be set before performing the first synchronization.

See also

- "truncate_timestamp_values option" on page 608
- "TIMESTAMP special value" [*SQL Anywhere Server - SQL Reference*]
- "UTC TIMESTAMP special value" [*SQL Anywhere Server - SQL Reference*]
- "default_timestamp_increment connection property" on page 625

delayed_commit_timeout option

Specifies the maximum delay between an application executing a COMMIT and the COMMIT actually being written to disk when the delayed_commits option is set to On.

Allowed values

Integer, in milliseconds

Default

500

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

This option has meaning only when delayed_commits is set to On. it governs when a COMMIT entry in the transaction log is written to disk. With delayed_commits set to On, the database server waits for the number of milliseconds set in the delayed_commit_timeout option for other connections to fill a page of the log before writing the current page contents to disk. See "delayed_commits option" on page 534.

See also

- "cooperative_commit_timeout option" on page 526
- "cooperative_commits option" on page 526
- "COMMIT statement" [*SQL Anywhere Server - SQL Reference*]
- "delayed_commit_timeout connection property" on page 625

delayed_commits option

Determines when the database server returns control to an application following a COMMIT.

Allowed values

On, Off

Default

Off (this corresponds to ISO COMMIT behavior)

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

When set to On, the database server replies to a COMMIT statement immediately instead of waiting until the transaction log entry for the COMMIT has been written to disk. When set to Off, the application must wait until the COMMIT is written to disk.

When this option is On, the log is written to disk when the log page is full or according to the delayed_commit_timeout option setting, whichever is first. There is a slight chance that a transaction may be lost even though committed if a system failure occurs after the database server replies to a COMMIT, but before the page is written to disk. Setting delayed_commits to On, and the delayed_commit_timeout option to a high value, promotes a quick response time at the slight risk of losing a committed transaction during recovery.

If both cooperative_commits and delayed_commits are set to On, and if the cooperative_commit_timeout interval passes without the pages getting written, the application is resumed (as if the commit had worked), and the remaining interval (delayed_commit_timeout - cooperative_commit_timeout) is used as a delayed_commits interval after which the pages will be written, even if they are not full.

See also

- "Use delayed commits" [*SQL Anywhere Server - SQL Usage*]
- "cooperative_commit_timeout option" on page 526
- "cooperative_commits option" on page 526
- "delayed_commit_timeout option" on page 534
- "COMMIT statement" [*SQL Anywhere Server - SQL Reference*]
- "delayed_commits connection property" on page 625

delete_old_logs option [SQL Remote]

Controls whether transaction logs are deleted when their transactions have been replicated or synchronized.

Allowed values

On, Off, Delay, *n* days

Default

Off

Remarks

This option is used by SQL Anywhere MobiLink clients and by SQL Remote. The default setting is Off. When it is set to On, each old transaction log is deleted when all the changes it contains have been replicated or synchronized successfully. When it is set to DELAY, each old transaction log with a file name indicating that it was created on the current day is not deleted, even if all changes have been sent and confirmed. When it is set to *n* days, logs that were created before *n* days ago are deleted.

For more information about how to use the delete_old_logs option in conjunction with the BACKUP statement to delete old copies of transaction logs, see "BACKUP statement" [*SQL Anywhere Server - SQL Reference*].

Example

If, on January 18 you run dbmlsync against a remote database that has set the delete_old_logs option to 10 days, dbmlsync deletes offline transaction logs that were created on or before January 7. The remote database would set the option as follows:

```
SET OPTION delete_old_logs = '10 days';
```

See also

- "Managing the transaction log" on page 919
- "SQL Anywhere client logging" [*MobiLink - Client Administration*]
- "Transaction Log utility (dblog)" on page 862
- MobiLink: "MirrorLogDirectory (mld) extended option" [*MobiLink - Client Administration*]
- SQL Remote: "Maintaining transaction logs for remote databases" [*SQL Remote*]
- "SQL Remote options" [*SQL Remote*]

divide_by_zero_error option

Controls the reporting of division by zero.

Allowed values

On, Off

Default

On

Remarks

This option indicates whether division by zero is reported as an error. If the option is set On, then division by zero results in an error with SQLSTATE 22012.

If the option is set Off, division by zero is not an error. Instead, a NULL is returned.

See also

- "divide_by_zero_error connection property" on page 626
- "Restrictions on materialized views" [*SQL Anywhere Server - SQL Usage*]

escape_character option

This option is reserved for system use. Do not change the setting of this option.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

exclude_operators option

This option is reserved for system use. Do not change the setting of this option.

extended_join_syntax option

Controls whether queries with duplicate correlation names syntax for multi-table joins are allowed, or reported as an error.

Allowed values

On, Off

Default

On

Remarks

If this option is set to On, then SQL Anywhere allows duplicate correlation names to be used in the null-supplying side of outer joins. All tables or views specified with the same correlation name are interpreted as the same instance of the table or view.

The following FROM clause illustrates the SQL Anywhere interpretation of a join using duplicate correlation names where C1 and C2 are search conditions:

```
( R left outer join T on ( C1 ), T  join S on ( C2 ) )
```

If the option is set to On, this join is interpreted as follows:

```
( R left outer join T on ( C1 ) ) join S on ( C2 )
```

If the option is set to Off, the following error is generated:

```
SQL Anywhere Error -137: Table 'T' requires a unique correlation name.
```

Note
To see the result of eliminating duplicate correlation names, you can view the rewritten statement using the REWRITE function with the second argument set to ANSI.

See also

- "REWRITE function [Miscellaneous]" [*SQL Anywhere Server - SQL Reference*]
- "Duplicate correlation names in joins (star joins)" [*SQL Anywhere Server - SQL Usage*]
- "extended_join_syntax connection property" on page 627

external_remote_options [SQL Remote]

Indicates where the message link parameters should be stored.

Allowed values

On, Off

Default

Off

Remarks

This option is used by SQL Remote to indicate whether the message link parameters should be stored in the database (Off) or externally (On).

See also

- "Set remote message type control parameters" [*SQL Remote*]

fire_triggers option

Controls whether triggers are fired in the database.

Allowed values

On, Off

Default

On

Remarks

When set to On, triggers are fired. When set to Off, no triggers are fired, including referential integrity triggers (such as cascading updates and deletes). Only a user with DBA authority can set this option. The option is overridden by the -gf option, which turns off all trigger firing regardless of the fire_triggers setting.

This option is relevant when replicating data from Adaptive Server Enterprise to SQL Anywhere because all actions from Adaptive Server Enterprise transaction logs are replicated to SQL Anywhere, including actions performed by triggers.

See also

- "-gf dbeng12/dbsrv12 server option" on page 187
- "Introduction to triggers" [*SQL Anywhere Server - SQL Usage*]
- "Compatibility options" on page 500
- "fire_triggers connection property" on page 627

first_day_of_week option

Sets the numbering of the days of the week.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Allowed values

1, 2, 3, 4, 5, 6, 7

Default

7 (Sunday is the first day of the week)

Remarks

The values have the following meaning:

Value	Meaning
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday
7	Sunday

The value specified by this option affects the result of the DATEPART function when obtaining a weekday value. You can also change the first day of week using the DATEFIRST option in the SET statement.

The value specified by this option does not affect the result of the DOW function. For example, even if the first day of the week is set to Monday, the DOW function returns a 2 for Monday.

See also

- "DATEPART function [Date and time]" [*SQL Anywhere Server - SQL Reference*]
- "DOW function [Date and time]" [*SQL Anywhere Server - SQL Reference*]
- "SET statement [T-SQL]" [*SQL Anywhere Server - SQL Reference*]
- "first_day_of_week connection property" on page 627

for_xml_null_treatment option

Controls the treatment of NULL values in queries that use the FOR XML clause.

Allowed values

Empty, Omit

Default

Omit

Remarks

If you execute a query that includes the FOR XML clause, the for_xml_null_treatment option determines how NULL values are treated. By default, elements and attributes that contain NULL values are omitted from the result. Setting this option to Empty generates empty elements or attributes if the value is NULL.

See also

- "Using the FOR XML clause to retrieve query results as XML" [*SQL Anywhere Server - SQL Usage*]
- "SELECT statement" [*SQL Anywhere Server - SQL Reference*]
- "for_xml_null_treatment connection property" on page 627

force_view_creation option

This option is reserved for system use. Do not change the setting of this option.

Caution
The force_view_creation option should only be used within a *reload.sql* script. This option is used by the Unload utility (dbunload) and should not be set explicitly.

global_database_id option

Controls the range of values for columns created with DEFAULT GLOBAL AUTOINCREMENT. For use in generating unique primary keys in a replication environment.

Allowed values

Non-negative integer

Default

2147483647

Scope

Can be set for the PUBLIC group only. DBA authority required.

Remarks

The value you specify for this option is the starting value. For columns created with DEFAULT GLOBAL AUTOINCREMENT, when a row is inserted into the table that does not include a value for the DEFAULT GLOBAL AUTOINCREMENT column, the database server generates a value for the column. The value is determined by the global_database_id value and the partition size for the column.

Setting global_database_id to the default value indicates that DEFAULT GLOBAL AUTOINCREMENT is disabled. In this case NULL is generated as a default.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can find the value of the option in the current database using the following statement:

```
SELECT DB_PROPERTY( 'GlobalDBID' );
```

This feature is of particular use in replication environments to ensure unique primary keys.

See also

- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "The GLOBAL AUTOINCREMENT default" [*SQL Anywhere Server - SQL Usage*]
- "Reloading tables with autoincrement columns" [*SQL Anywhere 12 - Changes and Upgrading*]
- "global_database_id connection property" on page 627
- MobiLink: "Setting the global database ID" [*MobiLink - Server Administration*]
- SQL Remote: "Duplicate primary key errors" [*SQL Remote*]

Example

The following example sets the database identification number to 100.

```
SET OPTION PUBLIC.global_database_id = '100';
```

http_connection_pool_basesize option

Specifies the nominal threshold size of database connections.

Allowed values

Integer, between 0 and 1000, inclusive

Default

10

Remarks

Changing this option takes effect within five seconds during the next purge cycle. Any unused database connections having exceeded http_connection_pool_timeout within the http_connection_pool_basesize threshold are deleted. When the size of a connection pool exceeds the http_connection_pool_basesize threshold, connections are deleted at a more aggressive rate.

Specifying a value of 0 purges the HTTP connection pools within five seconds of setting the option and disables HTTP connection pooling.

Unused excess connections are removed within half the time-out interval, half of the remainder are purged if unused within a quarter time-out interval and so on. The following database properties have been added to determine the operating efficiency of the connection pools within a database:

- Http_Conn_Pool_Cache_Count
- Http_Conn_Pool_Hits
- Http_Conn_Pool_Misses
- Http_Conn_Pool_Steals

See also

- "http_connection_pool_timeout option" on page 542
- "http_connection_pool_timeout connection property" on page 628
- "http_connection_pool_basesize connection property" on page 628
- "Database properties" on page 659
- "HTTP web services" [*SQL Anywhere Server - Programming*]

http_connection_pool_timeout option

Specifies the maximum duration that an unused connection can be retained in the connection pool.

Allowed values

Integer, between 1 and 86400, inclusive

Default

60

Remarks

Excess connections are deleted at an increasing rate based on this value if the pool size exceeds http_connection_pool_basesize.

Changing this option takes effect within five seconds during the next purge cycle. Any unused database connections having exceeded http_connection_pool_timeout within the http_connection_pool_basesize threshold are deleted. When the size of a connection pool exceeds the http_connection_pool_basesize threshold, connections are deleted at a more aggressive rate.

Unused excess connections are removed within half the time-out interval, half of the remainder are purged if unused within a quarter time-out interval and so on. The following database properties have been added to determine the operating efficiency of the connection pools within a database:

- Http_Conn_Pool_Cache_Count
- Http_Conn_Pool_Hits
- Http_Conn_Pool_Misses
- Http_Conn_Pool_Steals

See also

- "http_connection_pool_basesize option" on page 541
- "http_connection_pool_basesize connection property" on page 628
- "http_connection_pool_timeout connection property" on page 628
- "Database properties" on page 659
- "HTTP web services" [*SQL Anywhere Server - Programming*]

http_session_timeout option

Specify the default timeout duration, in minutes, that the HTTP session persists during inactivity.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Allowed values

Integer (1 to 525600)

Default

30

Scope

Can be set for the PUBLIC group only. DBA authority required.

Remarks

Setting this database option to another value sets a new default session timeout that is applied to all subsequently created HTTP sessions. An HTTP session can override this default by calling the sa_set_http_option system procedure to set the SessionTimeout within a particular HTTP session.

See also

- "sa_set_http_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Managing HTTP sessions on an HTTP server" [*SQL Anywhere Server - Programming*]
- "SessionCreateTime connection property" on page 639
- "SessionTimeout connection property" on page 639
- "http_session_timeout connection property" on page 628
- "HTTP web services" [*SQL Anywhere Server - Programming*]

integrated_server_name option

Specifies the name of the Domain Controller server used for looking up Windows user group membership for integrated logins.

Allowed values

String

Default

NULL

Scope

Can be set for the PUBLIC group only. DBA authority required.

Remarks

This option allows a user with DBA authority to specify the name of the Domain Controller server that is used to look up group membership when using Windows user groups for integrated logins. By default, the computer that SQL Anywhere is running on is used for verifying group membership.

See also

- “Creating integrated logins for Windows user groups” on page 112
- “GRANT statement” [*SQL Anywhere Server - SQL Reference*]
- “integrated_server_name connection property” on page 628

Example

The following example specifies that group membership is verified on the computer server-1.

```
SET OPTION PUBLIC.integrated_server_name = '\\server-1';
```

isolation_level option

Controls the locking isolation level.

Allowed values

0, 1, 2, 3, Snapshot, Statement-snapshot, Readonly-statement-snapshot

Default

0

1 for Sybase Open Client, jConnect, and TDS connections

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

This option controls the locking isolation level as follows:

- **0** Allow dirty reads, non-repeatable reads, and phantom rows.
- **1** Prevent dirty reads. Allow non-repeatable reads and phantom rows.
- **2** Prevent dirty reads and non-repeatable reads. Allow phantom rows.
- **3** Serializable. Prevent dirty reads, non-repeatable reads, and phantom rows.
- **Snapshot** Use a snapshot of committed data from the time when the first row is read or updated by the transaction.
- **Statement-snapshot** For each statement, use a snapshot of committed data from the time when the first row is read from the database. Non-repeatable reads and phantom rows can occur within a transaction, but not within a single statement.
- **Readonly-statement-snapshot** For read-only statements, use a snapshot of committed data from the time when the first row is read from the database. Non-repeatable reads and phantom rows can occur within a transaction, but not within a single statement. For updatable statements, use the

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

isolation level specified by the updatable_statement_isolation option (can be one of 0 (the default), 1, 2, or 3).

For more details about supported isolation levels, see "Isolation levels and consistency" [*SQL Anywhere Server - SQL Usage*].

The allow_snapshot_isolation option must be set to On to use the Snapshot, Statement-snapshot, or Readonly-statement-snapshot settings.

If you are using the SQL Anywhere JDBC driver, the default isolation level is 0.

Queries running at isolation level Snapshot, Statement-snapshot, or Readonly-statement-snapshot see a snapshot of a committed state of the database.

You can override any temporary or public settings for this option within individual INSERT, UPDATE, DELETE, SELECT, UNION, EXCEPT, and INTERSECT statements by including an OPTION clause in the statement.

See also

- "INSERT statement" [*SQL Anywhere Server - SQL Reference*]
- "UPDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "DELETE statement" [*SQL Anywhere Server - SQL Reference*]
- "SELECT statement" [*SQL Anywhere Server - SQL Reference*]
- "UNION statement" [*SQL Anywhere Server - SQL Reference*]
- "EXCEPT statement" [*SQL Anywhere Server - SQL Reference*]
- "INTERSECT statement" [*SQL Anywhere Server - SQL Reference*]
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]
- "allow_snapshot_isolation option" on page 506
- "updatable_statement_isolation option" on page 610
- "Snapshot isolation" [*SQL Anywhere Server - SQL Usage*]
- "Isolation levels and consistency" [*SQL Anywhere Server - SQL Usage*]
- "Choosing isolation levels" [*SQL Anywhere Server - SQL Usage*]
- "Compatibility options" on page 500
- "BEGIN TRANSACTION statement [T-SQL]" [*SQL Anywhere Server - SQL Reference*]
- "isolation_level connection property" on page 628

java_location option

Specifies the path of the Java VM for the database.

Allowed values

String

Default

Empty string

Scope

Can be set for the PUBLIC group only. DBA authority required.

Remarks

By default, this option contains an empty string. In this case, the database server searches the JAVA_HOME environment variable, the path, and other locations for the Java VM. The JavaVM database property allows you to query which Java VM the database server will use if the java_location option is not set.

See also

- "java_vm_options option" on page 546
- "Choosing a Java VM" [*SQL Anywhere Server - Programming*]
- "JavaVM database property" on page 669
- "java_location connection property" on page 628

java_vm_options option

Specifies command line options that the database server uses when it launches the Java VM.

Allowed values

String

Default

Empty string

Scope

Can be set for the PUBLIC group only. DBA authority required.

Remarks

This option lets you specify options that the database server uses when launching the Java VM specified by the java_location option. These additional options can be used to set up the Java VM for debugging purposes or to run as a service on Unix platforms. Sometimes additional options are required to use the Java VM in 64-bit mode instead of 32-bit mode.

See also

- "java_location option" on page 545
- "Choosing a Java VM" [*SQL Anywhere Server - Programming*]
- "java_vm_options connection property" on page 629

Example

The following example uses the java_vm_options option to keep the Java VM running on Unix when the database server is started as a service and the user needs to log out:

```
SET OPTION PUBLIC.java_vm_options = '-Xrs';
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The following example instructs the Java VM to use 64-bit mode on HP-UX:

```
SET OPTION PUBLIC.java_vm_options = '-d64';
```

log_deadlocks option

Controls whether deadlock reporting is turned on or off.

Allowed values

On, Off

Default

Off

Scope

Can be set for the PUBLIC group only. DBA authority required. Takes effect immediately.

Remarks

When this option is set to On, the database server logs information about deadlocks in an internal buffer. The size of the buffer is fixed at 10000 bytes. You can view the deadlock information using the sa_report_deadlocks stored procedure. The contents of the buffer are retained when this option is set to Off.

When deadlock occurs, information is reported for only those connections involved in the deadlock. The order in which connections are reported is based on which connection is waiting for which row. For thread deadlocks, information is reported about all connections.

When you have deadlock reporting turned on, you can also use the Deadlock system event to take action when a deadlock occurs. See "Understanding system events" on page 935.

You can also change the setting of this option by using the DeadlockLogging property with the sa_server_option system procedure.

See also

- "sa_report_deadlocks system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Determining who is blocked" [*SQL Anywhere Server - SQL Usage*]
- "Tutorial: Diagnosing deadlocks" [*SQL Anywhere Server - SQL Usage*]
- "log_deadlocks connection property" on page 630

login_mode option

Controls the use of integrated and Kerberos logins for the database.

Allowed values

One or more of: Standard, Integrated, Kerberos, Mixed (deprecated)

Default

Standard

Scope

Can be set for the PUBLIC group only. DBA authority required. Takes effect immediately.

Remarks

This option specifies whether standard, integrated, and Kerberos logins are permitted. One or more of the following login modes are accepted (the values are case insensitive):

- **Standard** Standard logins are permitted. This is the default setting. Standard connection logins must supply both a user ID and password, and do not use the Integrated or Kerberos connection parameters.
- **Integrated** Integrated logins are permitted.
- **Kerberos** Kerberos logins are permitted.
- **Mixed (deprecated)** This is equivalent to specifying Standard,Integrated.

If you specify multiple login modes, the database server allows all the specified modes.

Caution
Setting the login_mode database option to not allow Standard logins restricts connections to only those users or groups who have been granted an integrated or Kerberos login mapping. Attempting to connect with a user ID and password generates an error. The only exceptions to this are users with DBA authority.

You can specify multiple values in a comma-separated list. This list cannot contain white space. For example, the following setting allows both standard and integrated logins:

```
SET OPTION PUBLIC.login_mode = 'Standard,Integrated';
```

If a database file is not secured and can be copied by unauthorized users, the temporary public login_mode option should be used (both for integrated and Kerberos logins). This way, integrated and Kerberos logins are not supported by default if the file is copied.

See also

- "Using Windows integrated logins" on page 108
- "Kerberos authentication" on page 116
- "Controlling database access" on page 1117
- "Security concerns: Copied database files" on page 126
- "login_mode connection property" on page 630

Example

Enable only integrated logins (standard logins and Kerberos logins fail):

```
SET OPTION PUBLIC.login_mode = 'Integrated';
```

Enable standard and Kerberos logins (integrated logins fail):

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
SET OPTION PUBLIC.login_mode = 'Standard,Kerberos';
```

Enable standard, integrated, and Kerberos logins:

```
SET OPTION PUBLIC.login_mode = 'Standard,Integrated,Kerberos';
```

login_procedure option

Specifies a login procedure that sets connection compatibility options at startup.

Allowed values

String

Default

sp_login_environment system procedure

Scope

DBA authority required.

Remarks

This login procedure calls the sp_login_environment procedure at run time to determine the database connection settings. The login procedure is called after all the checks have been performed to verify that the connection is valid. The procedure specified by the login_procedure option is not executed for event connections, but it is executed for web service connections.

You can customize the default database option settings by creating a new procedure and setting login_procedure to call the new procedure. This custom procedure needs to call either sp_login_environment or detect when a TDS connection occurs (see the default sp_login_environment code) and call sp_tsql_environment directly. Failure to do so can break TDS-based connections. You should not edit either sp_login_environment or sp_tsql_environment.

A password expired error message with SQLSTATE 08WA0 can be signaled by a user defined login procedure to indicate to a user that their password has expired. Signaling the error allows applications to check for the error and process expired passwords. It is recommended that you use a login policy to implement password expiry and not a login procedure that returns the expired password error message.

If you use the NewPassword=***** connection parameter, signaling this error is required for the client libraries to prompt for a new password. If the procedure signals SQLSTATE 28000 (invalid user ID or password) or SQLSTATE 08WA0 (expired password), or the procedure raises an error with RAISERROR, the login fails and an error is returned to the user. If you signal any other error or if another error occurs, then the user login is successful and a message is written to the database server message log.

See also

- "post_login_procedure option" on page 571
- "sp_login_environment system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Increasing password security" on page 1118
- "NewPassword (NEWPWD) connection parameter" on page 299
- "CREATE PROCEDURE statement" [*SQL Anywhere Server - SQL Reference*]
- "login_procedure connection property" on page 630

Example

The following example shows how you can disallow a connection by signaling the INVALID_LOGON error.

```
CREATE PROCEDURE DBA.login_check( )
   BEGIN
      DECLARE INVALID_LOGON EXCEPTION FOR SQLSTATE '28000';
      // Allow a maximum of 3 concurrent connections
      IF( DB_PROPERTY( 'ConnCount' ) > 3 ) THEN
          SIGNAL INVALID_LOGON;
      ELSE
          CALL sp_login_environment;
      END IF;
   END
go

GRANT EXECUTE ON DBA.login_check TO PUBLIC
go

SET OPTION PUBLIC.login_procedure='DBA.login_check'
go
```

For more information about an alternate way to disallow connections, see "RAISERROR statement" [*SQL Anywhere Server - SQL Reference*].

The following example shows how you can block connection attempts if the number of failed connections for a user exceeds 3 within a 30 minute period. All blocked attempts during the block out period receive an invalid password error and are logged as failures. The log is kept long enough for a DBA to analyze it.

```
CREATE TABLE DBA.ConnectionFailure(
    pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
    user_name CHAR(128) NOT NULL,
    tm TIMESTAMP NOT NULL DEFAULT CURRENT TIMESTAMP
)
go

CREATE INDEX ConnFailTime ON DBA.ConnectionFailure(
    user_name, tm )
go

CREATE EVENT ConnFail TYPE ConnectFailed
HANDLER
BEGIN
    DECLARE usr CHAR(128);
    SET usr = event_parameter( 'User' );

    // Put a limit on the number of failures logged.
    IF (SELECT COUNT(*) FROM DBA.ConnectionFailure
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
        WHERE user_name = usr
        AND tm >= DATEADD( minute, -30,
            CURRENT TIMESTAMP )) < 20 THEN
        INSERT INTO DBA.ConnectionFailure( user_name )
            VALUES( usr );
        COMMIT;
        // Delete failures older than 7 days.
        DELETE DBA.ConnectionFailure
        WHERE user_name = usr
        AND tm < dateadd( day, -7, CURRENT TIMESTAMP );
        COMMIT;
    END IF;
END
go

CREATE PROCEDURE DBA.login_check( )
BEGIN
    DECLARE usr CHAR(128);
    DECLARE INVALID_LOGON EXCEPTION FOR SQLSTATE '28000';
    SET usr = CONNECTION_PROPERTY( 'Userid' );
    // Block connection attempts from this user
    // if 3 or more failed connection attempts have occurred
    // within the past 30 minutes.
    IF ( SELECT COUNT( * ) FROM DBA.ConnectionFailure
        WHERE user_name = usr
        AND tm >= DATEADD( minute, -30,
            CURRENT TIMESTAMP ) ) >= 3 THEN
        SIGNAL INVALID_LOGON;
    ELSE
        CALL sp_login_environment;
    END IF;
END
go

GRANT EXECUTE ON DBA.login_check TO PUBLIC
go

SET OPTION PUBLIC.login_procedure='DBA.login_check'
go
```

The following example shows how to signal the `Password has expired` message. It is recommended that you use a login policy to implement password expiry notification.

```
CREATE PROCEDURE DBA.check_expired_login( )
BEGIN
  DECLARE PASSWORD_EXPIRED EXCEPTION FOR SQLSTATE '08WA0';

  IF( condition-to-check-for-expired-password ) THEN
      SIGNAL PASSWORD_EXPIRED;
  ELSE
      CALL sp_login_environment;
  END IF;
END;
```

For information about login policies, see "Managing login policies" on page 471.

materialized_view_optimization option

Controls how materialized views are used by the optimizer to answer queries efficiently.

Allowed values

Disabled, **Fresh**, **Stale**, *N* { **Minute**[**s**] | **Hour**[**s**] | **Day**[**s**] | **Week**[**s**] | **Month**[**s**] }

Default

Stale

Scope

Can be set for an individual connection, for an individual user, or for the PUBLIC group. Takes effect immediately.

Remarks

The materialized_view_optimization option lets you specify the circumstances under which the optimizer can use stale materialized views.

Data in a materialized view becomes stale when data in any of the base tables referenced by the materialized view is updated. You should consider the acceptable degree of data staleness when deciding the refresh frequency for the materialized view, and the time it takes to refresh the view, since the view is not available for querying during the refresh process. You should also consider whether it is acceptable for the database server to return results that may not reflect the current state of the database. You can choose from the following settings for this option:

- **Disabled** Do not use materialized views for query optimization.
- **Fresh** Use a materialized view only if it is fresh (data in underlying tables has not been modified since the view was last refreshed).
- **Stale** Use materialized views even if they are stale. This is the default setting.
- **N { Minute[s] | Hour[s] | Day[s] | Week[s] | Month[s] }** Use fresh and stale materialized views, as long as the stale materialized views have been refreshed within the specified time period. Values specified in minutes must be less than 2^{31} minutes. The database server treats a week as 7 days and a month as 30 days.

When a query directly references a materialized view, the view is used regardless of staleness; the materialized_view_optimization option has no effect in this case.

See also

- “Improving performance with materialized views” [*SQL Anywhere Server - SQL Usage*]
- “Setting the optimizer staleness threshold for materialized views” [*SQL Anywhere Server - SQL Usage*]
- “materialized_view_optimization connection property” on page 630

max_client_statements_cached option

Controls the number of statements cached by the client.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Allowed values

Integer, 0 to 100

Scope

Can be set for an individual connection or for the PUBLIC group. Changing the value takes effect immediately.

Default

10

Remarks

Client statement caching reduces database requests and statement prepares when identical SQL statements are prepared multiple times. When the same SQL text is prepared and dropped repeatedly, the client caches the statement, leaving it prepared on the database server, even after it has been dropped by the application. Caching the statement saves the database server the extra work of dropping and re-preparing the statement. If a schema change occurs, a database option setting changes, or a DROP VARIABLE statement is executed, the prepared statement is dropped automatically and is prepared again the next time the SQL statement is executed, ensuring that a cached statement that could cause incorrect behavior is never reused.

This option specifies the maximum number of statements that can remain prepared (cached). Cached statements are not counted toward the max_statement_count resource governor.

The setting of this option applies to connections made using embedded SQL, ODBC, OLE DB, ADO.NET, and the SQL Anywhere JDBC driver. It does not apply to Sybase Open Client, jConnect, or HTTP connections.

Setting this option to 0 disables client statement caching. Increasing this value has the potential to improve performance if the application is repeatedly preparing and dropping more than ten of the same SQL statements. For example, if an application loops through twenty-five SQL statements, preparing and dropping them each iteration through the loop, and each iteration each of these SQL statements have the exact same text, setting this option to 25 may improve performance.

Increasing the value of this option increases memory use on the client and places more cache pressure on the database server. If a significant number of cached statements cannot be reused because of schema changes or option settings, statement caching is disabled automatically for the connection. If statement caching is automatically turned off, the client periodically turns statement caching on again to re-evaluate the decision and determine whether re-enabling statement caching would be beneficial.

When client statement caching is enabled, the max_client_statements_cached option should be set to 0 to disable client statement caching while the request log is captured if the log will be analyzed using the *tracetime.pl* Perl script.

See also

- “max_statement_count option” on page 557
- “ClientStmtCacheHits connection property” on page 623
- “ClientStmtCacheMisses connection property” on page 623
- “ClientStmtCacheHits server property” on page 646
- “ClientStmtCacheMisses server property” on page 646
- “max_client_statements_cached connection property” on page 630

max_cursor_count option

Controls a resource governor that limits the maximum number of cursors that a connection can use at once.

Allowed values

Integer

Default

50

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately. DBA authority required to set this option for any connection.

Remarks

This resource governor allows a DBA to limit the number of cursors per connection that a user can use. If an operation would exceed the limit for a connection, an error is generated, indicating that the governor for the resource has been exceeded.

If a connection executes a stored procedure, that procedure is executed under the permissions of the procedure owner. However, the resources used by the procedure are assigned to the current connection.

You can remove resource limits by setting the option to 0 (zero).

See also

- “Managing the resources connections use” on page 470
- “max_cursor_count connection property” on page 630

max_plans_cached option

Specifies the maximum number of execution plans to be stored in a cache.

Allowed values

Integer

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Default

20

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately. DBA authority required to set this option for the PUBLIC group.

Remarks

This option specifies the maximum number of plans cached for each connection. The optimizer caches the execution plan for queries, INSERT, UPDATE, and DELETE statements that are performed inside stored procedures, functions, and triggers. After a statement in a stored procedure, stored function, or trigger is executed several times by a connection, the optimizer builds a reusable plan for the statement.

Reusable plans do not use the values of host variables for selectivity estimation or rewrite optimizations. As a result of this, the reusable plan can have a higher cost than if the statement was re-optimized. When the cost of the reusable plan is close to the best observed cost for a statement, the optimizer adds the plan to the plan cache.

The cache is cleared when you execute statements, such as CREATE TABLE and DROP TABLE, that modify the table schema. Statements that reference declared temporary tables are not cached.

Setting this option to 0 disables plan caching.

See also

- "Plan caching" [*SQL Anywhere Server - SQL Usage*]
- "max_plans_cached connection property" on page 631

max_priority option

Controls the maximum priority level for connections.

Allowed values

Critical, High, Above Normal, Normal, Below normal, Low, Background

Default

Normal

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately. DBA authority required.

If you set this option temporarily, that setting applies to the current connection only. Different connections under the same user ID can have different settings for this option.

Remarks

The scheduling of different priority levels allows all requests to get some CPU time, regardless of the priority level of the request. Higher priority requests get more time slices than lower priority ones.

See also

- "Managing the resources connections use" on page 470
- "background_priority option [deprecated]" on page 515
- "priority option" on page 575
- "max_priority connection property" on page 631

max_query_tasks option

Specifies the maximum number of server tasks that the database server can use to process a query in parallel.

Allowed values

Integer

Default

0

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

The max_query_tasks option sets the maximum level of parallelism that can be used for any SQL statement. The option sets the number of database server tasks that can be used to process a query in parallel. The default value is 0, which allows the database server to use as many tasks as it chooses. Any other value for the max_query_tasks option sets the maximum number of tasks allowed per query. Setting the max_query_tasks option to 1 disables intra-query parallelism.

For more information about server tasks, threads, and query execution, see "Threading in SQL Anywhere" on page 49 and "Configuring the database server's multiprogramming level" on page 52.

The number of tasks the database server can use for all requests is limited by the threshold set using the -gn option at startup. This number is a global maximum for all databases and connections serviced by that server. The number of tasks used for a request is also limited by the number of logical processors available to the database server. For example, setting the processor concurrency to 1 with the -gtc option disables intra-query parallelism.

When enabled, intra-query parallelism is used to process SELECT statements that meet certain qualifications. The presence of an exchange operator in the access plan for a query indicates that intra-query parallelism was used.

You can override any temporary or public settings for this option within individual INSERT, UPDATE, DELETE, SELECT, UNION, EXCEPT, and INTERSECT statements by including an OPTION clause in the statement.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "-gn dbsrv12 server option" on page 189
- "-gt dbeng12/dbsrv12 server option" on page 195
- "-gtc dbeng12/dbsrv12 server option" on page 196
- "Parallelism during query execution" [*SQL Anywhere Server - SQL Usage*]
- "INSERT statement" [*SQL Anywhere Server - SQL Reference*]
- "UPDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "DELETE statement" [*SQL Anywhere Server - SQL Reference*]
- "SELECT statement" [*SQL Anywhere Server - SQL Reference*]
- "UNION statement" [*SQL Anywhere Server - SQL Reference*]
- "EXCEPT statement" [*SQL Anywhere Server - SQL Reference*]
- "INTERSECT statement" [*SQL Anywhere Server - SQL Reference*]
- "max_query_tasks connection property" on page 631

max_recursive_iterations option

Limits the maximum number of iterations a recursive common table expression can make.

Allowed values

Integer

Default

100

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately. DBA authority required to set this option for the PUBLIC group.

Remarks

Computation of a recursive common table expression aborts and an error is generated if the computation fails to complete within the specified number of iterations. Recursive subqueries often geometrically increase the amount of resources required for each additional iteration. Set this option to limit the amount of time and resources that will be consumed before infinite recursion is detected, yet permit your recursive common table expressions to work as intended.

Setting this option to 0 disables recursive common table expressions.

See also

- "Selecting hierarchical data" [*SQL Anywhere Server - SQL Usage*]
- "Restrictions on recursive common table expressions" [*SQL Anywhere Server - SQL Usage*]
- "max_recursive_iterations connection property" on page 631

max_statement_count option

Controls a resource governor that limits the maximum number of prepared statements that a connection can use simultaneously.

Allowed values

Integer

Default

50

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately. DBA authority required to set this option for any connection.

Remarks

Applications that use prepared statements can receive the error "Resource governor for 'prepared statements' exceeded" if the prepared statements are not explicitly dropped once they are no longer required. The max_statement_count database option is a resource governor that allows a DBA to limit the number of prepared statements used per connection. If an operation would exceed the limit for a connection, an error is generated, indicating that the governor for the resource has been exceeded.

If a connection executes a stored procedure, that procedure is executed under the permissions of the procedure owner. However, the resources used by the procedure are assigned to the current connection.

The database server maintains data structures for each prepared statement a connection creates. These structures are only freed when the application signals to the database server that the prepared statements are no longer needed or if the connection disconnects. To reduce the statement count for a connection, you must execute the equivalent of a DROP STATEMENT request. The following table lists the commands you can execute for the APIs supported by SQL Anywhere:

Interface	Statement
ADO	RecordSet.Close
ADO.NET	SADataReader.Close or SADataReader.Dispose
embedded SQL	DROP STATEMENT
Java	resultSet.Close, Statement.Close
ODBC	SQLFreeStmt(hstmt, SQL_DROP) or SQLFreeHandle(SQL_HANDLE_STMT, hstmt)

Note
In Java and .NET, it is recommended that you drop statements explicitly. You should not rely on garbage collection to perform this cleanup because the language routines do not issue server calls to deallocate the statement resources. In addition, there is no guarantee of when the garbage collection routines will execute.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

If a server needs to support more than the default number of prepared statements at any one time for any one connection, then the max_statement_count setting should be set to a higher value. Note, however, that larger numbers of active prepared statements consume additional server memory. You can disable the prepared statement resource governor entirely by setting the max_statement_count option to 0 (zero), but this is not recommended. Doing so makes the database server vulnerable to shutting down with an out-of-memory condition for applications that do not properly free prepared statements.

See also

- "Managing the resources connections use" on page 470
- "Preparing statements" [*SQL Anywhere Server - Programming*]
- "DROP STATEMENT statement [ESQL]" [*SQL Anywhere Server - SQL Reference*]
- "max_statement_count connection property" on page 631

max_temp_space option

Controls the maximum amount of temporary file space a connection can use.

Allowed values

Integer [**k** | **m** | **g** | **p**]

Default

0

Scope

Can be set for a temporary option for the duration of the current connection or for the PUBLIC group. Takes effect immediately. DBA authority required.

Remarks

This option allows you to specify the maximum amount of temporary file space a connection can use before the request fails because it exceeds the temporary file space limit. The temp_space_limit_check option must be set to On (the default) for the max_temp_space option to take effect.

The default value 0 indicates that there is no fixed limit on the amount of temporary file space a connection can request. Any other value specifies the number of bytes of temporary file space a connection can use. You can use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively. If you use **p**, the argument is a percentage of the total amount of temporary file space available.

For connections that request temporary file space, the database server checks the limit against the setting of the max_temp_space option to make sure the request is under the maximum size. If the connection requests more temporary space than is allowed, the request fails and the error SQLSTATE_TEMP_SPACE_LIMIT is generated.

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately. DBA authority required to set this option for the PUBLIC group.

See also

- "temp_space_limit_check option" on page 602
- "sa_disk_free_space system procedure" [*SQL Anywhere Server - SQL Reference*]
- "max_temp_space connection property" on page 631

Example

Set a 1 GB limit for a connection:

```
SET OPTION PUBLIC.max_temp_space = '1g';
```

Both of the following statements set a 1 MB limit for a connection:

```
SET OPTION PUBLIC.max_temp_space = 1048576;
SET OPTION PUBLIC.max_temp_space = '1m';
```

Use five percent of the total temporary space available:

```
SET OPTION PUBLIC.max_temp_space = '5p';
```

min_password_length option

Sets the minimum length for new passwords in the database.

Allowed values

Integer

The value is in bytes. For single-byte character sets, this is the same as the number of characters.

Default

0 characters

Scope

Can only be set for the PUBLIC group. Takes effect immediately. DBA authority required.

Remarks

This option allows the database administrator to impose a minimum length on all new passwords for greater security. Existing passwords are not affected. Passwords have a maximum length of 255 bytes and are case sensitive.

See also

- "verify_password_function option" on page 614
- "Increasing password security" on page 1118
- "min_password_length connection property" on page 631

Example

Set the minimum length for new passwords to 6 bytes.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
SET OPTION PUBLIC.min_password_length = 6;
```

nearest_century option

Controls the interpretation of two-digit years in string-to-date conversions.

Allowed values

Integer, between 0 and 100 inclusive

Default

50

Remarks

This option controls the handling of two-digit years when converting from strings to dates or timestamps.

The nearest_century setting is a numeric value that acts as a rollover point. Two digit years less than the value are converted to 20yy, while years greater than or equal to the value are converted to 19yy.

The historical SQL Anywhere behavior is to add 1900 to the year. Adaptive Server Enterprise behavior is to use the nearest century, so for any year where value *yy* is less than 50, the year is set to 20yy.

See also

- "Compatibility options" on page 500
- "Handling of two-digit years" [*SQL Anywhere Server - SQL Reference*]
- "Ambiguous date and time conversions" [*SQL Anywhere Server - SQL Reference*]
- "Converting dates to strings" [*SQL Anywhere Server - SQL Reference*]
- "nearest_century connection property" on page 632

non_keywords option

Turns off individual keywords, allowing their use as identifiers.

Allowed values

String

Default

Empty string

Remarks

This option turns off individual keywords. This provides a way of ensuring that applications created with older versions of the product are not broken by new keywords. If you have an identifier in your database that is now a keyword, you can either add double quotes around the identifier in all applications or scripts, or turn off the keyword using the non_keywords option.

You cannot turn off the keywords set, option, and options. Whether a word is identified as a keyword is determined by the following (in order of precedence):

- It appears in the SQL Anywhere list of reserved words.
- It has been turned on with the reserved_keywords option.
- It has been turned off using the non_keywords option.

The following statement prevents TRUNCATE and SYNCHRONIZE from being recognized as keywords:

```
SET OPTION non_keywords = 'TRUNCATE, SYNCHRONIZE';
```

Each new setting of this option replaces the previous setting. The following statement clears all previous settings.

```
SET OPTION non_keywords =;
```

A side-effect of this option is that SQL statements that use a turned off keyword cannot be used: they produce a syntax error.

You can turn on individual keywords using the reserved_keywords option. See "reserved_keywords option" on page 583.

See also

- "Keywords" [*SQL Anywhere Server - SQL Reference*]
- "Compatibility options" on page 500
- "non_keywords connection property" on page 632
- "reserved_keywords option" on page 583

odbc_describe_binary_as_varbinary

Controls how the SQL Anywhere ODBC driver describes BINARY columns.

Allowed values

On, Off

Default

Off

Remarks

This option allows you to choose whether you want all BINARY and VARBINARY columns to be described to your application as BINARY or VARBINARY. By default, the SQL Anywhere ODBC driver describes both BINARY and VARBINARY columns as SQL_BINARY. When this option is set to On, the ODBC driver describes BINARY and VARBINARY columns as SQL_VARBINARY. Regardless of the setting of this option, it is not possible to distinguish between BINARY and VARBINARY columns.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

It may be useful to turn this option On if you are using Delphi applications where BINARY columns are always zero-padded, but VARBINARY columns are not. You can improve performance in Delphi by setting this option to On so that all columns are treated as variable length data types.

See also

- "BINARY data type" [*SQL Anywhere Server - SQL Reference*]
- "VARBINARY data type" [*SQL Anywhere Server - SQL Reference*]
- "odbc_describe_binary_as_varbinary connection property" on page 633

odbc_distinguish_char_and_varchar option

Controls how the SQL Anywhere ODBC driver describes CHAR columns.

Allowed values

On, Off

Default

Off

Remarks

When a connection is opened, the SQL Anywhere ODBC driver uses the setting of this option to determine how CHAR columns are described. If this option is set to Off (the default), then CHAR columns are described as SQL_VARCHAR. If this option is set to On, then CHAR columns are described as SQL_CHAR. VARCHAR columns are always described as SQL_VARCHAR.

See also

- "CHAR data type" [*SQL Anywhere Server - SQL Reference*]
- "VARCHAR data type" [*SQL Anywhere Server - SQL Reference*]
- "odbc_distinguish_char_and_varchar connection property" on page 633

oem_string option

Stores user-specified information in the header page of the database file.

Allowed values

String (up to 128 bytes)

Default

Empty string

Scope

Can only be set for the PUBLIC group. Takes effect immediately. DBA authority required.

Remarks

You can store information in the header page of the database file and later extract the information by reading the file directly from your application. This page is stored in the system dbspace file header. If you specify a value for the OEM string that is longer than 128 bytes, an error is returned.

You may find it useful to store such information as schema versions, the application name, the application version, and so on. Alternatively, without starting the database, an application could use the OEM string to determine whether the database file is associated with the application, or design your application to use the information to validate that the database file is intended for your application, by storing a string that the application reads for validation purposes before using the database file. You could also extract metadata to display to users.

To set the oem_string in the system dbspace file header, execute the following statement:

```
SET OPTION PUBLIC.oem_string=user-specified-string;
```

The *user-specified-string* value is stored both in the ISYSOPTION system table and the system dbspace file header. You must define the string in the required character set before you specify it in a SET OPTION statement because no translation is done on the string when it is supplied in the SET OPTION statement. You can use the CSCONVERT function to convert the string to the required character set.

You can query the value of the oem_string in the following ways:

- Using the oem_string connection property:

  ```
  SELECT CONNECTION_PROPERTY( 'oem_string' );
  ```

- Using the SYSOPTION system view:

  ```
  SELECT setting FROM SYSOPTION WHERE "option" = 'oem_string';
  ```

To query the oem_string option from an application

1. Open the database system dbspace file.
2. Read the first page of the file into a buffer.
3. Search the buffer for the two byte prefix and suffix sequences before and after the OEM string.

   The prefix and suffix strings are defined in *sqldef.h* as DB_OEM_STRING_PREFIX and DB_OEM_STRING_SUFFIX, respectively. All the bytes between these two strings define the OEM string that is defined in the database.

SQL Anywhere includes two sample programs in the *oem_string* directory:

- *dboem.cpp* is a C program that illustrates how to extract the OEM string and print it to the database server messages window.
- *dboem.pl* illustrates how to extract the OEM string and print it to the stdout within a PERL script.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Caution
Applications cannot write directly to the OEM string in the database because it corrupts the database header page.

On Windows, applications cannot read the file directly when a server has the database file loaded. The database server has an exclusive lock on the file. However, on any supported Unix platform, applications that have read permissions can read the file directly at any time. However, changes to the OEM string may not show up in the file immediately. Issuing a checkpoint causes the database server to flush page 0 to disk, and reflect the current OEM string value.

Should the database server fail between changing the OEM string and the next checkpoint, the file header may not reflect the new OEM string value; the new OEM string value will be set correctly after the database goes through recovery.

See also

- "CSCONVERT function [String]" [*SQL Anywhere Server - SQL Reference*]
- "oem_string connection property" on page 633

Example

The following example encrypts the OEM string that contains information about the database file and stores it in the database header file:

```
BEGIN
  DECLARE @v VARCHAR(100);
  SET @v = BASE64_ENCODE( ENCRYPT( 'database version 10', 'abc' ) );
  EXECUTE IMMEDIATE 'SET OPTION PUBLIC.oem_string = ''' || @v || '''';
END;
```

You can retrieve the value of the OEM string using the following command:

```
SELECT DECRYPT(
   BASE64_DECODE(
      CONNECTION_PROPERTY( 'oem_string' ) ),'abc' )
```

on_charset_conversion_failure option

Controls what happens if an error is encountered during character conversion.

Allowed values

Ignore, Warning, Error

Default

Ignore

Remarks

Controls what happens if an error is encountered during character conversion, as follows:

- **Ignore** Errors and warnings do not appear.
- **Warning** Reports substitutions and illegal characters as warnings. Illegal characters are not translated.
- **Error** Reports substitutions and illegal characters as errors.

When character set conversion is required between the client and the database, this option governs whether to ignore, return a warning, or return an error, when illegal characters are detected, or when character substitution is used.

Single-byte to single-byte converters are not able to report substitutions and illegal characters, and must be set to Ignore.

This option does not control the behavior when lossy conversion takes place on the client. For example, SQL statements from the client must be in, or converted to, the CHAR database character set. Suppose a Unicode client application prepares a SQL statement, and that statement contains characters that cannot be represented in the CHAR database character set. Substitution characters are used instead. However, because the lossy conversion took place on the client, the database server is unaware of the lossy conversion.

See also

- "Comparisons between CHAR and NCHAR" [*SQL Anywhere Server - SQL Reference*]
- "Converting NCHAR to CHAR" [*SQL Anywhere Server - SQL Reference*]
- "Lossy conversion and substitution characters" [*SQL Anywhere Server - SQL Reference*]
- "on_charset_conversion_failure connection property" on page 633

on_tsql_error option

Controls error-handling in stored procedures.

Allowed values

- **Stop** Stop execution immediately upon finding an error.
- **Conditional** If the procedure uses ON EXCEPTION RESUME, and the statement following the error handles the error, continue, otherwise exit.
- **Continue** Continue execution, regardless of the following statement. If there are multiple errors, the first error encountered in the stored procedure is returned.

Default

Conditional

Continue for jConnect connections

Remarks

This option controls error handling in stored procedures.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Both the Conditional and Continue settings for on_tsql_error are used for Adaptive Server Enterprise compatibility, with Continue most closely simulating Adaptive Server Enterprise behavior. To have errors reported earlier, use the Conditional setting when creating new Transact-SQL stored procedures.

When this option is set to Stop or Continue, it supersedes the setting of the continue_after_raiserror option. However, when this option is set to Conditional (the default), behavior following a RAISERROR statement is determined by the setting of the continue_after_raiserror option.

See also

- "CREATE PROCEDURE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE PROCEDURE statement [T-SQL]" [*SQL Anywhere Server - SQL Reference*]
- "Transact-SQL procedure language overview" [*SQL Anywhere Server - SQL Usage*]
- "continue_after_raiserror option" on page 524
- "Compatibility options" on page 500
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]
- "on_tsql_error connection property" on page 633

optimization_goal option

Determines whether query processing is optimized towards returning the first row quickly, or minimizing the cost of returning the complete result set.

Allowed values

First-row, All-rows

Default

All-rows

Remarks

The optimization_goal option controls whether SQL Anywhere optimizes SQL data manipulation language (DML) statements for response time or total resource consumption.

If the option is set to All-rows (the default), then SQL Anywhere optimizes a query to choose an access plan with the minimal estimated total retrieval time. Setting optimization_goal to All-rows may be appropriate for applications that intend to process the entire result set, such as Sybase PowerBuilder DataWindow applications. A setting of All-rows is also appropriate for insensitive (ODBC static) cursors since the entire result is materialized when the cursor is opened. It may also be appropriate for scroll (ODBC keyset-driven) cursors, since the intent of such a cursor is to permit scrolling through the result set.

If the option is set to First-row, SQL Anywhere chooses an access plan that is intended to reduce the time to fetch the first row of the query's result, possibly at the expense of total retrieval time. In particular, the SQL Anywhere optimizer will typically avoid, if possible, access plans that require the materialization of results to reduce the time to return the first row. With this setting, the optimizer favors access plans that utilize an index to satisfy a query's ORDER BY clause, rather than plans that require an explicit sorting operation.

You can use the FASTFIRSTROW table hint in a query's FROM clause to set the optimization goal for a specific query to First-row, without having to change the optimization_goal setting.

For more information about using the FASTFIRSTROW table hint, see "FROM clause" [*SQL Anywhere Server - SQL Reference*].

You can override any temporary or public settings for this option within individual INSERT, UPDATE, DELETE, SELECT, UNION, EXCEPT, and INTERSECT statements by including an OPTION clause in the statement.

See also

- "INSERT statement" [*SQL Anywhere Server - SQL Reference*]
- "UPDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "DELETE statement" [*SQL Anywhere Server - SQL Reference*]
- "SELECT statement" [*SQL Anywhere Server - SQL Reference*]
- "UNION statement" [*SQL Anywhere Server - SQL Reference*]
- "EXCEPT statement" [*SQL Anywhere Server - SQL Reference*]
- "INTERSECT statement" [*SQL Anywhere Server - SQL Reference*]
- "optimization_goal connection property" on page 633

optimization_level option

Controls the amount of effort made by the SQL Anywhere query optimizer to find an access plan for a SQL statement.

Allowed values

0-15

Default

9

Remarks

The optimization_level option controls the amount of effort that the SQL Anywhere optimizer spends on optimizing SQL data manipulation language (DML) statements. This option controls the maximum number of alternative join strategies that the optimizer will consider for any SELECT block. The higher the setting of optimization_level, the greater the maximum number of join strategies that the optimizer will consider.

If the option is set to 0, then the SQL Anywhere optimizer chooses the first access plan it considers for execution, in effect avoiding any cost-based comparison of alternative plans. In addition, with level 0 some semantic optimizations of nested queries are disabled. If this option is set to a value higher than 0, the optimizer evaluates alternative strategies and chooses the one with the lowest expected cost. If this option is set to a value greater than the default (9), the optimizer is more aggressive in its search for alternative strategies, possibly resulting in much higher elapsed time spent in the optimization phase.

In typical scenarios, this option is temporarily set to lower levels (0, 1, or 2) when the application desires faster OPEN times for a DML statement. It is known that although the statement may be complex, the

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

query's execution time is very small, and the specific access plan chosen by the optimizer is less consequential. It is not recommended that the PUBLIC setting of optimization_level be changed from its default.

The effect of setting the optimization_level option is independent of the settings of the optimization_goal and optimization_workload options.

Simple DML statements (single-block, single-table queries that contain equality conditions in the WHERE clause that uniquely identify a specific row) are optimized heuristically and bypass the cost-based optimizer altogether. The optimization of simple DML statements is not affected by the setting of the optimization_level option. The count of the number of requests optimized through the optimizer bypass mechanism is available as the QueryBypassed connection property.

For more information about the QueryBypassed connection property, see "Connection properties" on page 619.

You can override any temporary or public settings for this option within individual INSERT, UPDATE, DELETE, SELECT, UNION, EXCEPT, and INTERSECT statements by including an OPTION clause in the statement.

See also

- "INSERT statement" [*SQL Anywhere Server - SQL Reference*]
- "UPDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "DELETE statement" [*SQL Anywhere Server - SQL Reference*]
- "SELECT statement" [*SQL Anywhere Server - SQL Reference*]
- "UNION statement" [*SQL Anywhere Server - SQL Reference*]
- "EXCEPT statement" [*SQL Anywhere Server - SQL Reference*]
- "INTERSECT statement" [*SQL Anywhere Server - SQL Reference*]
- "How the optimizer works" [*SQL Anywhere Server - SQL Usage*]
- "optimization_level connection property" on page 633

optimization_workload option

Determines whether query processing is optimized towards a workload that is a mix of updates and reads or a workload that is predominantly read-based.

Allowed values

Mixed, OLAP

Default

Mixed

Scope

Can be set for the PUBLIC group only. DBA authority required.

Remarks

The optimization_workload option controls whether SQL Anywhere optimizes queries for a workload that is a mix of updates and reads or that is predominantly read-only.

If the option is set to Mixed (the default), SQL Anywhere chooses query optimization algorithms appropriate for a workload that is a mixture of short inserts, updates, and deletes and longer running read-only queries.

If the option is set to OLAP, SQL Anywhere chooses algorithms appropriate for a workload that consists for the most part of long-running queries, combined with batch updates. In particular, the optimizer may choose to use the Clustered Hash Group By query execution algorithm.

When the option is set to OLAP, the Clustered Hash Group By algorithm is enabled. If the option is set to Mixed (the default), it is disabled.

You can override any temporary or public settings for this option within individual INSERT, UPDATE, DELETE, SELECT, UNION, EXCEPT, and INTERSECT statements by including an OPTION clause in the statement.

See also

- "INSERT statement" [*SQL Anywhere Server - SQL Reference*]
- "UPDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "DELETE statement" [*SQL Anywhere Server - SQL Reference*]
- "SELECT statement" [*SQL Anywhere Server - SQL Reference*]
- "UNION statement" [*SQL Anywhere Server - SQL Reference*]
- "EXCEPT statement" [*SQL Anywhere Server - SQL Reference*]
- "INTERSECT statement" [*SQL Anywhere Server - SQL Reference*]
- "ClusteredHashGroupBy algorithm (GrByHClust)" [*SQL Anywhere Server - SQL Usage*]
- "optimization_level option" on page 568
- "optimization_workload connection property" on page 633

pinned_cursor_percent_of_cache option

Specifies how much of the cache can be used for pinning cursors.

Allowed values

Integer, between 0-100

Default

10

Scope

Can be set for the PUBLIC group only. DBA authority required.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

The database server uses pages of virtual memory for the data structures needed to implement cursors. These pages are kept locked in memory between fetch requests so they are readily available when the next fetch request arrives.

To prevent these pages from occupying too much of the cache in low memory environments, a limit is placed on the percentage of the cache allowed to be used for pinning cursors. You can use the pinned_cursor_percent_of_cache option to adjust this limit.

The option value is specified as a percentage from 0 to 100, with a default of 10. Setting the option to 0 means that cursor pages are not pinned between fetch requests.

See also

- "pinned_cursor_percent_of_cache connection property" on page 634

post_login_procedure option

Specifies a procedure whose result set contains messages that should be displayed by applications when a user connects.

Allowed values

String

Default

post_login_procedure system procedure

Scope

DBA authority required.

Remarks

When the post_login_procedure option is set to anything other than an empty string, applications can call the procedure specified by the option as part of the connection process to determine what messages should be displayed to the user, if any. The option values should be of the form *owner*.*function-name* to prevent a user from overriding the function.

The SQL Anywhere 12 plug-in for Sybase Central and Interactive SQL call the procedure if this option is set and display any messages returned by the procedure in a window. Other applications that are not included with SQL Anywhere should be modified to call the procedure given by this option and display messages if you need this functionality.

One case where an application may need to display a message on connection is to notify the user that their password is about to expire if a password expiry system is implemented. This functionality could be used to notify the user each time they connect if their password will expire in the next few days, and before it actually expires.

The procedure specified by this option must return a result set with one or more rows and two columns. The first column of type VARCHAR(255) returns the text of the message, or NULL if there is no message. The second column of type INT returns the action type. Allowed values for actions are:

- **0** Display the message (if any).
- **1** Display the message and prompt the user for a password change.
- **2-99** Reserved.
- **100 and greater** User defined.

The SQL Anywhere 12 plug-in and Interactive SQL (dbisql) display all non-NULL messages, regardless of the action value. If the action is set to 1, then these tools prompt the user to change the password, and then set the new password to the user-specified value.

For an example that uses post_login_procedure and includes advanced password rules and implementing password expiration, see Using a password verification function.

See also

- "login_procedure option" on page 549
- "verify_password_function option" on page 614
- "Increasing password security" on page 1118
- "NewPassword (NEWPWD) connection parameter" on page 299
- "sa_post_login_procedure system procedure" [*SQL Anywhere Server - SQL Reference*]
- "post_login_procedure connection property" on page 634

Example

The following example uses a procedure named p_post_login_check that warns users that their password is about to expire and then prompts them to change their password.

```
CREATE PROCEDURE DBA.p_post_login_check( )
RESULT( message_text VARCHAR(255), message_action INT )
BEGIN
  DECLARE message_text        CHAR(255);
  DECLARE message_action     INT;

  -- assume the password_about_to_expire variable was
  -- set by the login procedure
  IF password_about_to_expire = 1 THEN
    SET message_text = 'Your password is about to expire';
    SET message_action = 1;
  ELSE
    SET message_text = NULL;
    SET message_action = 0;
  END IF;
  -- return message (if any) through this result set
  SELECT message_text, message_action;
END;

GRANT EXECUTE ON DBA.p_post_login_check TO PUBLIC;

SET OPTION PUBLIC.post_login_procedure = 'DBA.p_post_login_check';
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

precision option

Specifies the maximum number of digits in the result of any decimal arithmetic.

Allowed values

Integer, between 1 and 127, inclusive

Default

30

Scope

Can be set for the PUBLIC group only. Takes effect immediately.

Remarks

Precision is the total number of digits to the left and right of the decimal point. The scale option specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

Multiplication, division, addition, subtraction, and aggregate functions can all have results that exceed the maximum precision.

For example, when a DECIMAL(8,2) is multiplied with a DECIMAL(9,2), the result could require a DECIMAL(17,4). If precision is 15, only 15 digits will be kept in the result. If scale is 4, the result will be a DECIMAL(15,4). If scale is 2, the result will be a DECIMAL(15,2). In both cases, there is a possibility of overflow.

See also

- “scale option” on page 586
- “DECIMAL data type” [*SQL Anywhere Server - SQL Reference*]
- “NUMERIC data type” [*SQL Anywhere Server - SQL Reference*]
- “precision connection property” on page 634
- “scale connection property” on page 639

prefetch option

Controls whether rows are fetched to the client side before being made available to the client application.

Allowed values

Off, Conditional, Always

Default

Conditional

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

This option controls whether rows are fetched to the client side in advance of being made available to the client application. Fetching several rows at a time, even when the client application requests rows one at a time (for example, when looping over the rows of a cursor) can cut down on response time and improve overall throughput by cutting down the number of requests to the database.

- Off means no prefetching is done.
- Conditional (the default) causes prefetching to occur unless the cursor type is SENSITIVE or the query includes a proxy table.
- Always means prefetching is done even for sensitive cursor types and cursors that involve a proxy table.

The Always value must be used with caution, as it affects some cursor semantics. For example, it causes the sensitive cursor to become asensitive. Old values may be fetched if the value was updated between the prefetch and the application's fetch request. In addition, using prefetch on a cursor that involves a proxy table can cause the error -668, Cursor is restricted to FETCH NEXT operations, if the client attempts to re-fetch prefetch rows. A client may attempt to re-fetch prefetch rows after a rollback or on a fetch relative 0, if a fetch column is re-bound or bound for the first time after the first fetch, or when GET DATA is used.

The value sensitive cursor types include the ESQL SENSITIVE and SCROLL cursor types, and the ODBC and OLE DB DYNAMIC and KEYSET cursor types.

The setting of the prefetch option is ignored by Sybase Open Client and jConnect connections.

If the DisableMultiRowFetch connection parameter is set to YES, the prefetch database option is ignored and no prefetching is done.

This option previously accepted the value On. This value is now an alias for Conditional.

See also

- "Prefetching rows" [*SQL Anywhere Server - Programming*]
- "DisableMultiRowFetch (DMRF) connection parameter" on page 285
- "prefetch connection property" on page 634

preserve_source_format option

Controls whether the original source definition of procedures, triggers, views, and event handlers is saved in system files. If saved, it is saved in the column source in SYSTAB, SYSPROCEDURE, SYSTRIGGER, and SYSEVENT.

Allowed values

On, Off

Default

On

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Scope

Can be set for the PUBLIC group only. DBA authority required.

Remarks

When preserve_source_format is On, the database server saves the formatted source from CREATE and ALTER statements on procedures, views, triggers, and events, and puts it in the appropriate system view's source column.

Unformatted source text is stored in the same system tables, in the columns proc_defn, trigger_defn, and view_defn. However, these definitions are not easy to read in Sybase Central. The formatted source column allows you to view the definitions with the spacing, comments, and case that you want.

This option can be turned off to reduce space used to save object definitions in the database. The option can be set only for the user PUBLIC.

See also

- "preserve_source_format connection property" on page 634

prevent_article_pkey_update option

Controls updates to the primary key columns of tables involved in MobiLink publications.

Allowed values

On, Off

Default

On

Remarks

Setting this option to On disallows updates to the primary key columns of tables that are part of a publication. This option helps ensure data integrity, especially in a replication and synchronization environment.

> **Caution**
> It is strongly recommended that you do not set this option to Off in a synchronization or replication environment.

See also

- "prevent_article_pkey_update connection property" on page 634

priority option

Sets the priority level at which requests from a connection are executed.

Allowed values

Critical, High, Above Normal, Normal, Below Normal, Low, Background

Default

Normal

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

If you set this option temporarily, the setting applies to the current connection only. Different connections under the same user ID can have different settings for this option.

Remarks

The value of this option cannot be set higher than the value of the max_priority option.

See also

- "max_priority option" on page 555
- "BACKUP statement" [*SQL Anywhere Server - SQL Reference*]
- "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "REORGANIZE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "RESTORE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "UNLOAD statement" [*SQL Anywhere Server - SQL Reference*]
- "VALIDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "Managing the resources connections use" on page 470
- "priority connection property" on page 634

progress_messages option

Controls whether progress messages are sent from the database server to the client.

Allowed values

- **Off** Progress messages are not sent to the client.
- **Raw** When Raw is selected, the following format is used for progress messages:

  ```
  43;9728;22230;pages;5025;6138
  ```

  Raw progress messages have six fields separated by semicolons that are defined as follows:

  - **Field 1** The percentage of statement executed.
  - **Field 2** The number of completed pages, rows, or bytes.
  - **Field 3** The number of pages, rows, or bytes to be processed.
  - **Field 4** What is being processed: pages, rows, or bytes.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Field 5** The current elapsed time displayed in milliseconds.
- **Field 6** The estimated time in milliseconds remaining to complete the execution of the statement.

- **Formatted** When Formatted is selected, the following format is used for progress messages:

```
43% (9728 of 22230 pages) complete after 00:00:05; estimated 00:00:06
remaining
```

Formatted progress messages are localized, and the time format is HH:MM:SS. Units less than 100 KB are displayed in bytes, units less than 100 MB are displayed in KB, and units greater than 100 MB are displayed in MB.

Default

Off

Remarks

Progress messages are sent at intervals that are 5% of the total estimated duration of the statement. Typically, the estimate is completed and the first progress message is sent within 10 seconds. Additional progress messages are sent in intervals of 30 seconds to 5 minutes. If the percentage complete is identical to the value sent in a previous message, an updated progress message is not sent until more than 5 minutes have elapsed since the last message was sent. Progress messages are not sent for statements that take less than 30 seconds to execute.

Estimates are recalculated continually; the accuracy of the remaining time estimate increases as the operation progresses. During events such as backups, the total number of pages may be adjusted during statement execution, so the percent complete and remaining time estimates change. With statements such as BACKUP ... WITH CHECKPOINT COPY or UNLOAD SELECT the total number of affected pages or rows is unknown and it is possible for the percentage complete value to be greater than 100%. As a result, the estimated remaining time cannot be calculated and it is not included in the progress message.

The following statements and procedures support progress messages:

- BACKUP DATABASE (both image and archive)
- LOAD TABLE (USING FILE and USING CLIENT FILE only)
- MESSAGE
- REORGANIZE TABLE
- RESTORE DATABASE
- VALIDATE (all types)
- UNLOAD (all types)
- sa_table_page_usage system procedure

You can set the progress_messages option when you are connected to the utility database using the SET TEMPORARY OPTION statement. See “SET OPTION statement” [*SQL Anywhere Server - SQL Reference*], and “Allowed statements for the utility database” on page 29.

You can also sets the progress_messages option in Interactive SQL by choosing **Tools** » **Options** » **SQL Anywhere** » **Commands** and selecting **Show Progress Options**. When **Show Progress Options** is selected, the progress_messages option is set to Formatted.

See also

- "BACKUP statement" [*SQL Anywhere Server - SQL Reference*]
- "Backup utility (dbbackup)" on page 767
- "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "MESSAGE statement" [*SQL Anywhere Server - SQL Reference*]
- "REORGANIZE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "RESTORE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "UNLOAD statement" [*SQL Anywhere Server - SQL Reference*]
- "VALIDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "Progress connection property" on page 635

qualify_owners option [SQL Remote]

Controls whether SQL statements being replicated by SQL Remote should use qualified object names.

Allowed values

On, Off

Default

On

Remarks

When qualification is not needed in SQL Anywhere installations, messages will be slightly smaller with this option set to Off.

See also

- "SQL Remote options" [*SQL Remote*]

query_mem_timeout option

Sets the maximum time, in milliseconds, that a request waits for a memory grant.

Allowed values

-1, 0, positive integer

Default

-1

Remarks

When this option is set to -1 (the default) or any value less than 0, the request waits for a memory grant for up to 50 times the estimated execution time for the request. If this option is set to 0, the request waits forever for memory to be granted. Otherwise, the value is the maximum time, in milliseconds, that a request waits for a memory grant.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "The memory governor" [*SQL Anywhere Server - SQL Usage*]
- "query_mem_timeout connection property" on page 635

quote_all_identifiers option [SQL Remote]

Controls whether SQL statements being replicated by SQL Remote should use quoted identifiers.

Allowed values

On, Off

Default

Off

Remarks

When this option is Off, dbremote quotes identifiers that require quotes by SQL Anywhere (as it has always done).

When the option is On, all identifiers are quoted.

See also

- "SQL Remote options" [*SQL Remote*]

quoted_identifier option

Controls the interpretation of strings that are enclosed in double quotes.

Allowed values

On, Off

Default

On

Off for Sybase Open Client and jConnect connections

Remarks

This option controls whether strings that are enclosed in double quotes are interpreted as identifiers (On) or as literal strings (Off). The quoted_identifier option is included for Transact-SQL compatibility.

See "Setting options for Transact-SQL compatibility" [*SQL Anywhere Server - SQL Usage*].

See also

- "Compatibility options" on page 500
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]
- "SET statement [T-SQL]" [*SQL Anywhere Server - SQL Reference*]
- "quoted_identifier connection property" on page 636

read_past_deleted option

Controls database server behavior on uncommitted deletes at isolation levels 1 and 2.

Allowed values

On, Off

Default

On

Remarks

If read_past_deleted is On (the default), sequential scans at isolation levels 1 and 2 skip uncommitted deleted rows. If Off, sequential scans block on uncommitted deleted rows at isolation levels 1 and 2 (until the deleting transaction commits or rolls back). This option changes server behavior at isolation levels 1 and 2.

For most purposes, this option should be left On. If set to Off, the blocking behavior depends on the plan chosen by the optimizer (if there is an index that could possibly be used).

See also

- "Isolation levels and consistency" [*SQL Anywhere Server - SQL Usage*]
- "read_past_deleted connection property" on page 636

recovery_time option

Sets the maximum length of time, in minutes, that the database server will take to recover from system failure.

Allowed values

Integer, in minutes

Default

2

Scope

Can be set for the PUBLIC group only. DBA authority required. Takes effect when server is restarted.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

This option is used with the checkpoint_time option to decide when checkpoints should be done.

SQL Anywhere uses a heuristic to estimate the recovery time based on the operations that have been performed since the last checkpoint, and includes both the estimated recovery time and the estimated checkpoint time for the database. So, the recovery time is not exact.

See also

- "The automatic recovery process" on page 902
- "checkpoint_time option" on page 519
- "-gr dbeng12/dbsrv12 server option" on page 194
- "How the database server decides when to checkpoint" on page 924
- "recovery_time connection property" on page 636

remote_idle_timeout option

Controls how many seconds of inactivity web service client procedures and functions tolerate.

Allowed values

Integer, in seconds

Default

15

Remarks

This option affects web service client procedures and functions. If more time than the specified number of seconds passes without activity, the procedure or function times out.

See also

- "Developing web client applications" [*SQL Anywhere Server - Programming*]
- "remote_idle_timeout connection property" on page 637
- "CREATE PROCEDURE statement (web clients)" [*SQL Anywhere Server - SQL Reference*]
- "CREATE FUNCTION statement (web clients)" [*SQL Anywhere Server - SQL Reference*]

replication_error option [SQL Remote]

Allows you to specify a stored procedure to be called by the Message Agent when a SQL error occurs.

Allowed values

Stored procedure name

Default

No procedure

Remarks

For SQL Remote, the replication_error option allows you to specify a stored procedure to be called by the Message Agent when a SQL error occurs. By default, no procedure is called.

The procedure must have a single argument of type CHAR, VARCHAR, or LONG VARCHAR. The procedure is called once with the SQL error message and once with the SQL statement that causes the error. In some circumstances (such as foreign key violations), the SQL statement that caused the error is not available, so the stored procedure can only be called once.

Although the option allows you to track and monitor SQL errors in replication, you must still design them out of your setup; this option is not intended to resolve such errors.

See also

- "SQL Remote options" [*SQL Remote*]
- "replication_error_piece option [SQL Remote]" on page 582
- "Run an error-handling procedure" [*SQL Remote*]

replication_error_piece option [SQL Remote]

Works in conjunction with the replication_error option to allow you to specify a LONG VARCHAR stored procedure to be called by the Message Agent when a SQL error occurs during SQL Remote replication.

Allowed values

Stored procedure name

Default

No procedure

Remarks

If an error occurs and replication_error is defined, then the replication_error procedure is called with the full error string.

If replication_error and replication_error_piece are both defined, then the error is broken up into VARCHAR pieces. replication_error is called with the first piece and replication_error_piece is called repeatedly with the remaining pieces.

See also

- "replication_error option [SQL Remote]" on page 581
- "SQL Remote options" [*SQL Remote*]

request_timeout option

Controls the maximum time a single request can run. This option can be used to prevent a connection from consuming a significant amount of server resources for a long period of time.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Allowed values

Integer, 0 through 86400 (one day), in seconds

Default

0

Remarks

When this option is set to 0, requests do not time out.

Any request that takes longer than approximately request_timeout seconds (wall-clock time, not CPU time) is interrupted and an error is returned to the user. The error returned is SQLE_REQUEST_TIMEOUT: "Request interrupted due to timeout". If a request is blocked, and the blocking_timeout option is set to 0, then the request can remain blocked for a maximum of request_timeout seconds before returning a blocking error (for example, SQLE_LOCKED: "User '%1' has the row in '%2' locked").

Values in the range 1 to 14 cannot be specified as a user option or the PUBLIC option, but can be specified as a temporary option. This prevents users from being locked out of the database server if connecting takes a long time (for example, because of a complex login procedure).

This option can be used with both database client and HTTP/HTTPS requests. Note that setting the option in a stored procedure or HTTP/HTTPS request has no effect on the current request since the option value at the beginning of the request is used.

Setting the request_timeout public option should be done with caution as this can cause applications that have long running requests (such as dbvalid, dbbackup, and dbunload) to fail. Also, applications that do not use significant server resources, but that can block on another user can fail when request_timeout is set. One way to address these types of problems is to set the request_timeout option only for certain applications in the login procedure based on a connection's APPINFO value.

Setting this option may not prevent applications from using significant server resources if each request evaluates quickly, for example when fetching a result set containing many rows.

See also

- “blocking_timeout option” on page 517
- “AppInfo (APP) connection parameter” on page 266
- “request_timeout connection property” on page 638

reserved_keywords option

Turns on individual keywords disabled by default.

Allowed values

String

Default

Empty string

Remarks

This option turns on individual keywords that are disabled by default. Currently, only the LIMIT keyword can be turned on.

The following statement allows the LIMIT keyword to be recognized as a keyword:

```
SET OPTION reserved_keywords = 'LIMIT';
```

You cannot turn on the keywords set, option, and options. Whether a word is identified as a keyword is determined by the following (in order of precedence):

- It appears in the SQL Anywhere list of reserved words.
- It has been turned on with the reserved_keywords option.
- It has been turned off using the non_keywords option.

Each new setting of this option replaces the previous setting. The following statement clears all previous settings.

```
SET OPTION reserved_keywords =;
```

You can turn off individual keywords using the non_keywords option. See "non_keywords option" on page 561.

See also

- "Keywords" [*SQL Anywhere Server - SQL Reference*]
- "non_keywords option" on page 561
- "Reserved words" [*SQL Anywhere Server - SQL Reference*]

return_date_time_as_string option

Controls how a date, time, or timestamp value is passed to the client application when queried.

Allowed values

On, Off

Default

Off

Scope

Can be set as a temporary option only, for the duration of the current connection.

Remarks

This option indicates whether date, time, and timestamp values are returned to applications as a date or time data type or as a string.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

When this option is set to On, the database server converts the date, time, or timestamp value to a string before it is sent to the client to preserve the timestamp_format, date_format, or time_format option setting.

Sybase Central and Interactive SQL automatically turn the return_date_time_as_string option On.

See also

- "date_format option" on page 528
- "time_format option" on page 603
- "timestamp_format option" on page 604
- "timestamp_with_time_zone_format option" on page 606
- "return_date_time_as_string connection property" on page 639

rollback_on_deadlock

Controls how transactions are treated when a deadlock occurs.

Allowed values

On, Off

Default

On

Scope

Can be set by any user and can be set for the PUBLIC group and individual connections. Takes effect immediately.

Remarks

When this option is set to On, a transaction is automatically rolled back if it encounters a deadlock. The rollback happens after the current request completes. If this option is set to Off, SQL Anywhere automatically rolls back the statement that encountered the deadlock, and returns an error to that transaction indicating which form of deadlock occurred. Note that rolling back the statement likely would not release any of the locks acquired by the statement.

See also

- "Deadlock" [*SQL Anywhere Server - SQL Usage*]
- "rollback_on_deadlock connection property" on page 639

row_counts option

Specifies whether the database will always count the number of rows in a query when it is opened.

Allowed values

On, Off

Default

Off

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

If this option is set to Off, the row count is usually only an estimate. If this option is set to On, the row count is always accurate.

> **Caution**
> When row_counts is set to On, it may take significantly longer to execute queries. In fact, it will usually cause SQL Anywhere to execute the query twice, doubling the execution time.

See also

- C API: "sqlany_num_rows method" [*SQL Anywhere Server - Programming*]
- PHP API: "sasql_num_rows" [*SQL Anywhere Server - Programming*]
- "row_counts connection property" on page 639

save_remote_passwords option [SQL Remote]

Saves the password that is entered in the message link.

Allowed values

On, Off

Default

On

Remarks

If you are storing the message link parameters externally, rather than in the database, you may not want to save the passwords. You can prevent the passwords from being saved by setting this option to Off.

See also

- "SQL Remote options" [*SQL Remote*]

scale option

Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Allowed values

Integer, between 0 and 127, inclusive, and less than the value specified for the precision database option

Default

6

Scope

Can be set for the PUBLIC group only. Takes effect immediately.

Remarks

Multiplication, division, addition, subtraction, and aggregate functions can all have results that exceed the maximum precision.

See also

- "precision option" on page 573
- "scale connection property" on page 639
- "precision connection property" on page 634

secure_feature_key

Allows you to enable features for the connection that were secured using the database server -sf option.

Allowed values

String

Default

NULL

Scope

Can be set as a temporary option only, for the duration of the current connection.

Remarks

You can specify features that cannot be used by databases running on a server by including the -sf option when you start the database server. The -sk server option lets you specify a key that can be used to re-enable all secured (disabled) features for a connection and gives that connection authority to change the features that are secured for all databases running on the database server. When you set the value of the secure_feature_key temporary option to the value specified by -sk when the database server was started, then all features are re-enabled for that database connection, and on that connection you can use the sa_server_option system procedure to control access to database features.

If the secure_feature_key option is set to any value other than the one specified by -sk, no error is given, and the features specified by -sf remain disabled for the connection.

See also

- "-sk dbeng12/dbsrv12 server option" on page 224
- "-sf dbeng12/dbsrv12 server option" on page 219
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Specifying secured features" on page 1122
- "secure_feature_key connection property" on page 639

Example

The following command starts a database server named secure_server with access to the request log and all remote data access features disabled. The key specified by the -sk option can be used later to enable these features for a specific database connection.

```
dbsrv12 -n secure_server -sf request_log,remote -sk j978kls12 testdb.db
```

Setting the secure_feature_key option to the value specified by -sk for a database running on the secure_server database server enables access to the request log and remote data access features for that connection:

```
SET TEMPORARY OPTION secure_feature_key = 'j978kls12';
```

sort_collation option

Allows implicit use of the SORTKEY function on ORDER BY expressions.

Allowed values

Internal, collation_name, or collation_id

Default

Internal

Remarks

When the value of this option is Internal, the ORDER BY clause remains unchanged.

When the value of this option is set to a valid collation name or collation ID, CHAR or NCHAR string expressions in the ORDER BY clause are treated as if the SORTKEY function had been invoked. String expressions that use other string data types, such as BINARY, UUID, XML, or VARBIT are not modified.

See also

- "SORTKEY function [String]" [*SQL Anywhere Server - SQL Reference*]
- "sort_collation connection property" on page 640

Example

Set the sort collation to binary:

```
SET TEMPORARY OPTION sort_collation='binary';
```

Having the sort collation set to binary transforms the following queries:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
SELECT Name, ID
FROM Products
ORDER BY Name, ID;
SELECT name, ID
FROM Products
ORDER BY 1, 2;
```

The queries are transformed into:

```
SELECT Name, ID
FROM Products
ORDER BY SORTKEY(Name, 'binary'), ID;
```

sql_flagger_error_level option

Controls the response to any SQL that is not part of the specified standard.

Allowed values

- Off
- SQL:1992/Entry
- SQL:1992/Intermediate
- SQL:1992/Full
- SQL:1999/Core
- SQL:1999/Package
- SQL:2003/Core
- SQL:2003/Package
- UltraLite

Default

Off

Remarks

This option flags as an error any SQL that is not part of the specified standard. For example, specifying SQL:2003/Package causes the database server to flag syntax that is not full SQL/2008 syntax.

The default behavior, Off, turns error flagging off.

For compatibility with previous SQL Anywhere versions, the following values are also accepted, and are mapped as specified below:

- **E** This option corresponds to SQL:1992/Entry.
- **I** This option corresponds to SQL:1992/Intermediate.
- **F** This option corresponds to SQL:1992/Full.
- **W** This option corresponds with Off.

See also

- "sa_ansi_standard_packages system procedure" [*SQL Anywhere Server - SQL Reference*]
- "SQLFLAGGER function [Miscellaneous]" [*SQL Anywhere Server - SQL Reference*]
- "sql_flagger_warning_level option" on page 590
- "SQL preprocessor" [*SQL Anywhere Server - Programming*]
- "Compatibility options" on page 500
- "sql_flagger_error_level connection property" on page 640

sql_flagger_warning_level option

Controls the response to any SQL that is not part of the specified standard.

Allowed values

- Off
- SQL:1992/Entry
- SQL:1992/Intermediate
- SQL:1992/Full
- SQL:1999/Core
- SQL:1999/Package
- SQL:2003/Core
- SQL:2003/Package
- UltraLite

Default

Off

Remarks

This option flags any SQL that is not part of a specified standard as a warning. For example, specifying SQL:2003/Package causes the database server to flag syntax that is not full SQL/2008 syntax.

The default behavior, Off, turns warning flagging off.

For compatibility with previous versions, the following values are also accepted, and are mapped as specified below:

- **E** This option corresponds to SQL:1992/Entry.
- **I** This option corresponds to SQL:1992/Intermediate.
- **F** This option corresponds to SQL:1992/Full.
- **W** This option corresponds to Off.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "sa_ansi_standard_packages system procedure" [*SQL Anywhere Server - SQL Reference*]
- "SQLFLAGGER function [Miscellaneous]" [*SQL Anywhere Server - SQL Reference*]
- "sql_flagger_error_level option" on page 589
- "SQL preprocessor" [*SQL Anywhere Server - Programming*]
- "Compatibility options" on page 500
- "sql_flagger_warning_level connection property" on page 640

sr_date_format option [SQL Remote]

Sets the format for dates retrieved from the database.

Allowed values

String (composed of the symbols listed below)

Default

yyyy/mm/dd

Remarks

The Message Agent uses this option when replicating columns that store a date. The format is a string using the following symbols:

Symbol	Description
YY	Two digit year
YYYY	Four digit year
MM	Two digit month)
MMM[m...]	Character short form for months—as many characters as there are "m"s
DD	Two digit day of month

Each symbol is substituted with the appropriate data for the date that is being formatted.

For symbols that represent character data (such as *MMM*), you can control the case of the output as follows:

- Type the symbol in all uppercase to have the format appear in all uppercase. For example, MMM produces JAN.
- Type the symbol in all lowercase to have the format appear in all lowercase. For example, mmm produces jan.

- Type the symbol in mixed case to have SQL Anywhere choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.

If the character data is multibyte, the length of each symbol reflects the number of characters, not the number of bytes. For example, the 'mmm' symbol specifies a length of three characters for the month.

For symbols that represent numeric data, you can control zero-padding with the case of the symbols:

The option is a string build from the following symbols:

- Type the symbol in same-case (such as MM or mm) to allow zero padding. For example, yyyy/mm/dd could produce 2002/01/01.
- Type the symbol in mixed case (such as Mm) to suppress zero padding. For example, yyyy/Mm/Dd could produce 2002/1/1.

See also

- "sr_time_format option [SQL Remote]" on page 592
- "sr_timestamp_format [SQL Remote]" on page 593
- "sr_timestamp_with_time_zone_format [SQL Remote]" on page 593
- "SQL Remote options" [*SQL Remote*]

sr_time_format option [SQL Remote]

Sets the format for times retrieved from the database.

Allowed values

String (composed of the symbols listed below)

Default

hh:nn:ss.Ssssss

Remarks

The Message Agent uses this option when replicating columns that store a time. The format is a string using the following symbols:

Symbol	Description
HH	Two digit hours (24-hour clock).
NN	Two digit minutes.
MM	Two-digit minutes if following a colon (as in HH:MM).

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Symbol	Description
SS.SSSSSS	Seconds and fractions of a second, up to six decimal places. Not all platforms support timestamps to a precision of six places.

Each symbol is substituted with the appropriate data for the time that is being formatted. Any format symbol that represents character rather than digit output can be put in uppercase, which causes the substituted characters to also be in uppercase. For numbers, using mixed case in the format string suppresses leading zeros.

Remarks

Using mixed case in the formatting string suppresses leading zeros.

See also

- "sr_date_format option [SQL Remote]" on page 591
- "sr_timestamp_format [SQL Remote]" on page 593
- "sr_timestamp_with_time_zone_format [SQL Remote]" on page 593
- "SQL Remote options" [*SQL Remote*]

sr_timestamp_format [SQL Remote]

Sets the format for timestamps that are retrieved from the database.

Default

yyyy/mm/dd hh:nn:ss.Ssssss

Remarks

The Message Agent replicates datetime and timestamp information using this option.

See also

- "sr_date_format option [SQL Remote]" on page 591
- "sr_time_format option [SQL Remote]" on page 592
- "sr_timestamp_with_time_zone_format [SQL Remote]" on page 593
- "SQL Remote options" [*SQL Remote*]

sr_timestamp_with_time_zone_format [SQL Remote]

Sets the format for timestamp with time zone values that are retrieved from the database.

Default

yyyy/mm/dd hh:nn:ss.Ssssss +hh:nn

Remarks

The Message Agent replicates timestamp with time zone information using this option.

See also

- "sr_date_format option [SQL Remote]" on page 591
- "sr_time_format option [SQL Remote]" on page 592
- "sr_timestamp_format [SQL Remote]" on page 593
- "SQL Remote options" [*SQL Remote*]

st_geometry_asbinary_format option

Controls how spatial values are converted from a geometry to binary.

Allowed values

The list of values supported for this option is identical to the list of parameters (including sub-parameters) documented for the ST_AsBinary method. See "ST_AsBinary method for type ST_Geometry" [*SQL Anywhere Server - Spatial Data Support*].

Default

WKB

Remarks

This option is used to determine how a geometry is converted to binary. The following are the two main contexts in which the option is used:

- fetching a value as binary from a client application
- executing a CAST statement. For example: `CAST( geometry-expression AS LONG BINARY )`

Assignments from geometry types to BINARY types give an error in contexts other than fetching from a client application and the CAST statement. For example, you cannot call a function or procedure that takes a BINARY parameter and pass a geometry, nor can you assign a geometry to a LONG BINARY variable.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "ST_AsBinary method for type ST_Geometry" [*SQL Anywhere Server - Spatial Data Support*]
- "st_geometry_astext_format option" on page 595
- "st_geometry_asxml_format option" on page 596
- "st_geometry_describe_type option" on page 597
- "st_geometry_on_invalid option" on page 598
- "st_geometry_asbinary_format connection property" on page 640
- "st_geometry_astext_format connection property" on page 640
- "st_geometry_asxml_format connection property" on page 641
- "st_geometry_describe_type connection property" on page 641
- "st_geometry_on_invalid connection property" on page 641

st_geometry_astext_format option

Controls how spatial values are converted from a geometry to text.

Allowed values

The list of values supported for this option is identical to the list of parameters (including sub-parameters) documented for the ST_AsText method. See "ST_AsText method for type ST_Geometry" [*SQL Anywhere Server - Spatial Data Support*].

Default

WKT

Remarks

This option is used to determine how a geometry is converted to text. The following are the two main contexts in which the option is used:

- fetching a value as text from a client application
- Executing a CAST statement. For example: `CAST( geometry-expression AS LONG VARCHAR )`

Assignments from geometry types to CHAR types give an error in contexts other than fetching from a client application and the CAST statement. For example, you cannot call a function or procedure that takes a CHAR parameter and pass a geometry, nor can you assign a geometry to a LONG VARCHAR variable.

See also

- "ST_AsText method for type ST_Geometry" [*SQL Anywhere Server - Spatial Data Support*]
- "st_geometry_asbinary_format option" on page 594
- "st_geometry_asxml_format option" on page 596
- "st_geometry_describe_type option" on page 597
- "st_geometry_on_invalid option" on page 598
- "st_geometry_asbinary_format connection property" on page 640
- "st_geometry_astext_format connection property" on page 640
- "st_geometry_asxml_format connection property" on page 641
- "st_geometry_describe_type connection property" on page 641
- "st_geometry_on_invalid connection property" on page 641

st_geometry_asxml_format option

Controls how spatial values are converted from a geometry to XML.

Allowed values

The list of values supported for this option is identical to the list of parameters (including sub-parameters) documented for the ST_AsXML method. See "ST_AsXML method for type ST_Geometry" [*SQL Anywhere Server - Spatial Data Support*].

Default

GML

Remarks

This option is used to determine how a geometry is converted to XML. The following are the two main contexts in which the option is used:

- fetching a value as XML from a client application
- Executing a CAST statement. For example: `CAST( geometry-expression AS XML )`

Assignments from geometry types to XML give an error in contexts other than fetching from a client application and the CAST statement. For example, you cannot call a function or procedure that takes an XML parameter and pass a geometry, nor can you assign a geometry to an XML variable.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "ST_AsXML method for type ST_Geometry" [*SQL Anywhere Server - Spatial Data Support*]
- "st_geometry_asbinary_format option" on page 594
- "st_geometry_astext_format option" on page 595
- "st_geometry_describe_type option" on page 597
- "st_geometry_on_invalid option" on page 598
- "st_geometry_asbinary_format connection property" on page 640
- "st_geometry_astext_format connection property" on page 640
- "st_geometry_asxml_format connection property" on page 641
- "st_geometry_describe_type connection property" on page 641
- "st_geometry_on_invalid connection property" on page 641

st_geometry_describe_type option

Controls how spatial values are described.

Allowed values

- CHAR
- NCHAR
- BINARY

Default

CHAR

Remarks

Spatial values are fetched as character or binary because client libraries do not support spatial types directly. The st_geometry_describe_type specifies how a geometry column or expression is described to the client. Many client applications use the describe type for a column or expression to determine the client data type used to fetch the column or expression. Some client applications ignore the describe type, so setting the st_geometry_describe_type may not have the desired affects.

If a geometry is fetched as a character string type, the st_geometry_astext_format option is used to determine how to convert the geometry to text. If fetched as a binary type, the st_geometry_asbinary_format option is used. These are the same rules used by a CAST statement.

In ODBC applications, the SQL_DESC_TYPE_NAME and SQL_DESC_LOCAL_TYPE_NAME field identifiers of SQLColAttribute return the string `st_geometry` for columns of any geometry type. In JDBC, this is expressed by using the ResultSetMetaData.getColumnTypeName method.

See also

- "st_geometry_asbinary_format option" on page 594
- "st_geometry_astext_format option" on page 595
- "st_geometry_asxml_format option" on page 596
- "st_geometry_on_invalid option" on page 598
- "st_geometry_asbinary_format connection property" on page 640
- "st_geometry_astext_format connection property" on page 640
- "st_geometry_asxml_format connection property" on page 641
- "st_geometry_describe_type connection property" on page 641
- "st_geometry_on_invalid connection property" on page 641

st_geometry_on_invalid option

Controls the behavior when a geometry fails basic validation (for example, a linestring with 1 point, a polygon with a ring that is not closed, or a geometry that exceeds the bounds of the spatial reference system).

Allowed values

- **Ignore** Do nothing when a geometry fails surface validation, and continue with the operation.
- **Error** Return an error when a geometry fails surface validation, and fail the operation.

Default

Error

Remarks

If you need to load invalid spatial data, you can set the option value to Ignore and then load the invalid data. Once loaded, you can use methods such as ST_IsSimple and ST_IsValue to determine the invalid row, ST_AsText to get the WKT representation of the data. Most other spatial methods give an error if they are passed invalid geometries. Use a text editor to correct the invalid geometry in the WKT, and then update the geometry in the database.

See also

- "st_geometry_asbinary_format option" on page 594
- "st_geometry_astext_format option" on page 595
- "st_geometry_asxml_format option" on page 596
- "st_geometry_describe_type option" on page 597
- "st_geometry_asbinary_format connection property" on page 640
- "st_geometry_astext_format connection property" on page 640
- "st_geometry_asxml_format connection property" on page 641
- "st_geometry_describe_type connection property" on page 641
- "st_geometry_on_invalid connection property" on page 641

string_rtruncation option

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Determines whether an error is raised when a string is truncated.

Allowed values

On, Off

Default

On

Remarks

If the truncated characters consist only of spaces, no exception is raised. The setting of On corresponds to ANSI/ISO SQL/2008 behavior. When this option is set to Off, the exception is not raised and the character string is silently truncated.

String truncation may occur in several places. For example, using INSERT, UPDATE, CAST, or assignment to a variable may truncate a string if the declared destination type is too short.

See also

- "Character data types" [*SQL Anywhere Server - SQL Reference*]
- "Compatibility options" on page 500
- "SET statement [T-SQL]" [*SQL Anywhere Server - SQL Reference*]
- "string_rtruncation connection property" on page 641

subscribe_by_remote option [SQL Remote]

Controls the interpretation of NULL or empty-string SUBSCRIBE BY values.

Allowed values

On, Off

Default

On

Remarks

When the option is set to On, operations from remote databases on rows with a SUBSCRIBE BY value that is NULL or an empty string assume that the remote user is subscribed to the row. When it is set to Off, the remote user is assumed not to be subscribed to the row.

The only limitation of this option is that it will lead to errors if a remote user really does want to INSERT (or UPDATE) a row with a NULL or empty subscription expression (for information held only at the consolidated database). This is reasonably obscure and can be worked around by assigning a subscription value in your installation that belongs to no remote user.

See also

- "SQL Remote options" [*SQL Remote*]
- "Using the subscribe_by_remote option with many-to-many relationships" [*SQL Remote*]

subsume_row_locks option

Controls when the database server acquires individual row locks for a table.

Allowed values

On, Off

Default

On

Remarks

If the subsume_row_locks option is On (the default) then whenever a table *t* is locked exclusively with LOCK TABLE *t* IN EXCLUSIVE MODE, the database server no longer acquires individual row locks for *t*.

This can result in a significant performance improvement if extensive updates are made to *t* in a single transaction, especially if *t* is large relative to cache size. It also allows for atomic update operations that are larger than the lock table can currently handle (approximately 2-4 million rows).

When this option is On, keyset cursors over a table locked in this fashion will return row changed warnings for every row in the cursor, if any row in the database has been modified. Note that the database server could turn an updatable cursor with an ORDER BY into a keyset cursor as a result.

See also

- "LOCK TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "subsume_row_locks connection property" on page 641

suppress_tds_debugging option

Determines whether TDS debugging information appears in the database server messages window.

Allowed values

On, Off

Default

Off

Remarks

When the database server is started with the -z option, debugging information appears in the database server messages window, including debugging information about the TDS protocol.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The suppress_tds_debugging option restricts the debugging information about TDS that appears in the database server messages window. When this option is set to Off (the default) TDS debugging information appears in the database server messages window.

See also

- "-z dbeng12/dbsrv12 server option" on page 243
- "suppress_tds_debugging connection property" on page 641

synchronize_mirror_on_commit option

Controls when database changes are assured to have been sent to a mirror server when running in asynchronous or asyncfullpage mode.

Allowed values

On, Off

Default

Off

Remarks

The synchronize_mirror_on_commit option allows fine-grained control over when database changes are assured to have been sent to a mirror server when running in asynchronous or asyncfullpage mode. The option is Off by default. When set to On, each COMMIT causes any changes recorded in the transaction log to be sent to the mirror server, and an acknowledgement to be sent by the mirror server to the primary server once the changes are received by the mirror server. The option can be set for specific transactions using SET TEMPORARY OPTION. It may also be useful to set the option for specific applications by examining the APPINFO string in a login procedure. This allows mirroring behavior to be tailored to meet the needs of different applications.

See also

- "Introduction to database mirroring" on page 945
- "synchronize_mirror_on_commit connection property" on page 641

tds_empty_string_is_null option

Controls whether empty strings are returned as NULL or a string containing one blank character for TDS connections.

Allowed values

On, Off

Default

Off

Remarks

By default, this option is set to Off and empty strings are returned as a string containing one blank character for TDS connections. When this option is set to On, empty strings are returned as NULL strings for TDS connections. Non-TDS connections distinguish empty strings from NULL strings.

See also

- "NULL value" [*SQL Anywhere Server - SQL Reference*]
- "tds_empty_string_is_null connection property" on page 641

temp_space_limit_check option

Checks the amount of temporary file space used by a connection and fails the request if the amount of space requested is greater than the connection's allowable quota.

Allowed values

On, Off

Default

On

Scope

Can be set for the PUBLIC group only. DBA authority required.

Remarks

When temp_space_limit_check is set to On (the default), if a connection requests more than its quota of temporary file space, then the request fails and the error SQLSTATE_TEMP_SPACE_LIMIT is returned. When this option is set to Off, the database server does not check the amount of temporary file space used by a connection. If a connection requests more than its quota of temporary space when this option is set to Off, a fatal error can occur.

The temporary file space quota for a connection is the minimum of the following two thresholds:

1. the maximum amount of temporary file space permitted for each connection as specified by the setting of the max_temp_space option

2. the maximum potential size of the temporary file divided by the number of connections

This threshold is used only if the temporary file has grown to 80% or more of its maximum size, which is determined by the amount of free space remaining on the device as reported by the operating system. When a connection requests more temporary file space than the quota allows, that connection's current request fail with SQLSTATE 54W05 (TEMP_SPACE_LIMIT).

You can specify a hard limit on the amount of temporary file space used by a connection with the max_temp_space option.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "sa_disk_free_space system procedure" [*SQL Anywhere Server - SQL Reference*]
- "max_temp_space option" on page 559
- "-dt dbeng12/dbsrv12 server option" on page 175
- "temp_space_limit_check connection property" on page 641

time_format option

Sets the format for times retrieved from the database.

Allowed values

String (composed of the symbols listed below)

Default

HH:NN:SS.SSS

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

The format is a string using the following symbols:

Symbol	Description
HH	Two digit hours
NN	Two digit minutes
SS.SSSSSS	Seconds and fractions of a second, up to six decimal places. Not all platforms support timestamps to a precision of six places.
AA	A.M. or P.M. (12 hour clock)—omit AA and PP for 24 hour time
PP	PM if needed (12 hour clock)—omit AA and PP for 24 hour time

Each symbol is substituted with the appropriate data for the time that is being formatted. Any format symbol that represents character rather than digit output can be put in uppercase, which causes the substituted characters to also be in uppercase. For numbers, using mixed case in the format string suppresses leading zeros.

See also

- “date_format option” on page 528
- “timestamp_format option” on page 604
- “timestamp_with_time_zone_format option” on page 606
- “Compatibility options” on page 500
- “sp_tsql_environment system procedure” [*SQL Anywhere Server - SQL Reference*]
- “Retrieving dates and times from the database” [*SQL Anywhere Server - SQL Reference*]
- “time_format connection property” on page 642

time_zone_adjustment option

Allows a connection's time zone adjustment to be modified.

Allowed values

Unsigned integer (for example, 300)

A positive or negative signed integer enclosed in quotation marks (for example, '+300' or '-300')

String representing a time in hours and minutes, preceded by + or - and enclosed in quotation marks (for example, '+5:00', or '-5:00')

Default

If the client is connecting via embedded SQL, ODBC, OLE DB, ADO, or ADO.NET, the default value is set according to the client's time zone. If the client is connecting via jConnect or Sybase Open Client, the default is based on the database server's time zone.

Remarks

The time_zone_adjustment option value is the same value as that returned by `SELECT CONNECTION_PROPERTY( 'TimeZoneAdjustment' );`. The value represents the number of minutes that must be added to the Coordinated Universal Time (UTC) to display time local to the connection.

For information on how this option controls conversion of TIMESTAMP to TIMESTAMP WITH TIME ZONE, see “Converting to or from TIMESTAMP WITH TIME ZONE” [*SQL Anywhere Server - SQL Reference*].

See also

- “time_zone_adjustment connection property” on page 642

timestamp_format option

Sets the format for timestamps that are retrieved from the database.

Allowed values

String (composed of the symbols listed below)

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Default

YYYY-MM-DD HH:NN:SS.SSS

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

The format is a string using the following symbols:

Symbol	Description
YY	Two digit year
YYYY	Four digit year
MM	Two digit month, or two digit minutes if following a colon (as in HH:MM)
MMM[m...]	Character short form for months—as many characters as there are "m"s
DD	Two digit day of month
DDD[d...]	Character short form for day of the week
HH	Two digit hours
NN	Two digit minutes
SS.SSSSSS	Seconds and fractions of a second, up to six decimal places. Not all platforms support timestamps to a precision of six places.
AA	A.M. or P.M. (12 hour clock)—omit AA and PP for 24 hour time
PP	PM if needed (12 hour clock)—omit AA and PP for 24 hour time

Each symbol is substituted with the appropriate data for the date that is being formatted.

For symbols that represent character data (such as *MMM*), you can control the case of the output as follows:

- Type the symbol in all uppercase to have the format appear in all uppercase. For example, MMM produces JAN.
- Type the symbol in all lowercase to have the format appear in all lowercase. For example, mmm produces jan.
- Type the symbol in mixed case to have SQL Anywhere choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.

If the character data is multibyte, the length of each symbol reflects the number of characters, not the number of bytes. For example, the 'mmm' symbol specifies a length of three characters for the month.

For symbols that represent numeric data, you can control zero-padding with the case of the symbols:

- Type the symbol in same-case (such as MM or mm) to allow zero padding. For example, yyyy/mm/dd could produce 2002/01/01.
- Type the symbol in mixed case (such as Mm) to suppress zero padding. For example, yyyy/Mm/Dd could produce 2002/1/1.

Note
If you change the setting for timestamp_format in a way that re-orders the date format, be sure to change the date_order option to reflect the same change, and vice versa. See “date_order option” on page 530.

See also

- “date_format option” on page 528
- “time_format option” on page 603
- “timestamp_with_time_zone_format option” on page 606
- “Compatibility options” on page 500
- “sp_tsql_environment system procedure” [*SQL Anywhere Server - SQL Reference*]
- “timestamp_format connection property” on page 642

timestamp_with_time_zone_format option

Sets the format for TIMESTAMP WITH TIME ZONE values retrieved from the database.

Allowed values

String (composed of the symbols listed below)

Default

YYYY-MM-DD HH:NN:SS.SSS+HH:NN

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

The format is a string using the following symbols:

Symbol	Description
YY	Two digit year
YYYY	Four digit year

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Symbol	Description
MM	Two digit month, or two digit minutes if following a colon (as in HH:MM)
MMM[m...]	Character short form for months—as many characters as there are "m"s
DD	Two digit day of month
DDD[d...]	Character short form for day of the week
HH	Two digit hours
NN	Two digit minutes
SS.SSSSSS	Seconds and fractions of a second, up to six decimal places. Not all platforms support timestamps to a precision of six places.
AA	A.M. or P.M. (12 hour clock)—omit AA and PP for 24 hour time
PP	P.M. if needed (12 hour clock)—omit AA and PP for 24 hour time
HH	Two digit hours (time zone offset)
NN	Two digit minutes (time zone offset)

Each symbol is substituted with the appropriate data for the date that is being formatted.

For symbols that represent character data (such as *MMM*), you can control the case of the output as follows:

- Type the symbol in all uppercase to have the format appear in all uppercase. For example, MMM produces JAN.
- Type the symbol in all lowercase to have the format appear in all lowercase. For example, mmm produces jan.
- Type the symbol in mixed case to have SQL Anywhere choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.

If the character data is multibyte, the length of each symbol reflects the number of characters, not the number of bytes. For example, the mmm symbol specifies a length of three characters for the month.

For symbols that represent numeric data, you can control zero-padding with the case of the symbols:

- Type the symbol in same-case (such as MM or mm) to allow zero padding. For example, yyyy/mm/dd could produce 2002/01/01.
- Type the symbol in mixed case (such as Mm) to suppress zero padding. For example, yyyy/Mm/Dd could produce 2002/1/1.

Note
If you change the setting for timestamp_with_time_zone_format option in a way that re-orders the date format, be sure to change the date_order option to reflect the same change, and vice versa. See "date_order option" on page 530.

See also

- "TIMESTAMP WITH TIME ZONE data type" [*SQL Anywhere Server - SQL Reference*]
- "date_format option" on page 528
- "time_format option" on page 603
- "timestamp_format option" on page 604

truncate_timestamp_values option

Limits the resolution of timestamp values.

Allowed values

On, Off

Default

Off

Scope

Can be set for the PUBLIC group only. DBA authority required. This option should not be enabled for databases already containing timestamp data.

Remarks

A TIMESTAMP value is precise to six decimal places in SQL Anywhere. However, to maintain compatibility with other software, which may truncate the TIMESTAMP value to three decimal places, you can set the truncate_timestamp_values option to On to limit the number of decimal places SQL Anywhere stores. The default_timestamp_increment option determines the number of decimal places to which the TIMESTAMP value is truncated.

For MobiLink synchronization, if you are going to set this option, it must be set before performing the first synchronization.

If the database server finds TIMESTAMP values with a higher resolution than that specified by the combination of truncate_timestamp_values and default_timestamp_increment, an error is reported.

Unloading the database and then reloading it into a new database in which the truncate_timestamp_values and default_timestamp_increment values have been set is usually the easiest solution to ensure the proper TIMESTAMP values are used. However, depending on the type of TIMESTAMP columns in your table, you can also do the following:

- If the TIMESTAMP columns are defined with DEFAULT TIMESTAMP or DEFAULT UTC TIMESTAMP (so that the value is automatically updated by the database server when the row is

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

modified), you must delete all the rows in the table before the truncate_timestamp_values option is changed. You can delete the rows using the DELETE or TRUNCATE TABLE statement.

- If the TIMESTAMP column is defined with a value other than DEFAULT TIMESTAMP or DEFAULT UTC TIMESTAMP, you can execute an UPDATE statement that casts the values to a string and then back to a TIMESTAMP. For example,

  ```
  UPDATE T
     SET ts = CAST( DATEFORMAT( ts, 'yyyy/mm/dd hh:nn:ss.ss' )
   AS TIMESTAMP );
  ```

  Note that this process may lose more precision than is necessary. The format string to use depends on the number of digits of precision to be kept.

See also

- "default_timestamp_increment option" on page 533
- "TIMESTAMP special value" [*SQL Anywhere Server - SQL Reference*]
- "CURRENT UTC TIMESTAMP special value" [*SQL Anywhere Server - SQL Reference*]
- "UTC TIMESTAMP special value" [*SQL Anywhere Server - SQL Reference*]
- "truncate_timestamp_values connection property" on page 642

Example

Setting the default_timestamp_increment option to 100000 causes truncation after the first decimal place in the seconds component, allowing a value such as '2000/12/05 10:50:53:700' to be stored.

tsql_outer_joins option

Controls the ability to use the Transact-SQL outer join operators *= and =* in statements and views.

Allowed values

On, Off

Default

Off

Remarks

Support for Transact-SQL outer joins is deprecated. Setting this option to On allows you to use Transact-SQL outer joins.

See also

- "Compatibility options" on page 500
- "Transact-SQL outer joins (*= or =*)" [*SQL Anywhere Server - SQL Usage*]
- "Join operators" [*SQL Anywhere Server - SQL Reference*]
- "tsql_outer_joins connection property" on page 642

tsql_variables option

Controls whether the @ sign can be used as a prefix for embedded SQL host variable names.

Allowed values

On, Off

Default

Off

On for Sybase Open Client and jConnect connections

Remarks

When this option is set to On, you can use the @ sign instead of the colon as a prefix for host variable names in embedded SQL. This is implemented primarily for Transact-SQL compatibility.

See also

- "Compatibility options" on page 500
- "sp_tsql_environment system procedure" [*SQL Anywhere Server - SQL Reference*]
- "tsql_variables connection property" on page 642

updatable_statement_isolation option

Specifies the isolation level used by updatable statements when the isolation_level option is set to Readonly-statement-snapshot.

Allowed values

0, 1, 2, 3

Default

0

Remarks

The isolation level specified by the updatable_statement_isolation option is used by updatable statements when the isolation_level option is set to Readonly-statement-snapshot. The following values are accepted:

- **0** Allow dirty reads, non-repeatable reads, and phantom rows.
- **1** Prevent dirty reads. Allow non-repeatable reads and phantom rows.
- **2** Prevent dirty reads and non-repeatable reads. Allow phantom rows.
- **3** Serializable. Prevent dirty reads, non-repeatable reads, and phantom rows.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "isolation_level option" on page 544
- "Snapshot isolation" [*SQL Anywhere Server - SQL Usage*]
- "Isolation levels and consistency" [*SQL Anywhere Server - SQL Usage*]
- "Choosing isolation levels" [*SQL Anywhere Server - SQL Usage*]
- "Typical types of inconsistency" [*SQL Anywhere Server - SQL Usage*]
- "updatable_statement_isolation connection property" on page 642

update_statistics option

Controls whether connections can send query feedback to the statistics governor.

Allowed values

On, Off

Default

On

Remarks

The database server collects statistics during normal query execution and uses the gathered statistics to self-tune the column statistics. When you set this option to Off, the statistics governor does not receive feedback about the health of statistics or fix any statistics for the specified connection. However, it does continue to monitor the usage of statistics and create or drop statistics based on their usage.

When this option is set by a user with DBA authority, its setting affects all connections to the database; otherwise, only the current users connections are affected.

Under normal circumstances, it should not be necessary to turn this option off.

The update_statistics option does not affect changes to statistics as a result of updates to data (LOAD/INSERT/UPDATE/DELETE). To control whether statistics are updated based on these statements, use the collect_statistics_on_dml_updates database option.

The difference between the collect_statistics_on_dml_updates option and the update_statistics option is that the update_statistics option compares the actual number of rows that satisfy a predicate with the number of rows that are estimated to satisfy the predicate, and then updates the estimates. The collect_statistics_on_dml_updates option modifies the column statistics based on the values of the specific rows that are inserted, updated, or deleted.

See also

- "collect_statistics_on_dml_updates option" on page 521
- "How the statistics governor maintains statistics" [*SQL Anywhere Server - SQL Usage*]
- "Updating column statistics to improve optimizer performance" [*SQL Anywhere Server - SQL Usage*]
- "update_statistics connection property" on page 642

user_estimates option

Controls whether user selectivity estimates in query predicates are respected or ignored by the query optimizer.

Allowed values

Enabled, Disabled, Override-Magic

Default

Override-Magic

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

SQL Anywhere allows you to specify user selectivity estimates can improve the optimizer's performance when the database server is unable to accurately predict the selectivity of a predicate. However, user selectivity estimates should be used only in appropriate circumstances. For example, it may be useful to supply a selectivity estimate for a predicate that involves one or more functions if the Override-Magic selectivity estimate used by the optimizer is significantly different from the actual selectivity.

If you have used selectivity estimates that are inaccurate as a workaround to performance problems where the software-selected access plan was poor, it is recommended that you set this option to Disabled. The database server may not select an optimal plan if you use inaccurate estimates.

For more information about user selectivity estimates, see "Explicit selectivity estimates" [*SQL Anywhere Server - SQL Reference*].

When a user selectivity estimate is supplied with a predicate, the estimate is respected or ignored based on the setting of this option. The following values are accepted:

- **Enabled** All user-supplied selectivity estimates are respected. You can also use On to turn on this option.
- **Override-Magic** A user selectivity estimate is respected and used only if the optimizer would otherwise choose to use its last-resort, heuristic value (also called the magic value).
- **Disabled** All user estimates are ignored and magic values are used when no other estimate data is available. You can also use Off to turn off this option.

You can override any temporary or public settings for this option within individual INSERT, UPDATE, DELETE, SELECT, UNION, EXCEPT, and INTERSECT statements by including an OPTION clause in the statement.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "INSERT statement" [*SQL Anywhere Server - SQL Reference*]
- "UPDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "DELETE statement" [*SQL Anywhere Server - SQL Reference*]
- "SELECT statement" [*SQL Anywhere Server - SQL Reference*]
- "UNION statement" [*SQL Anywhere Server - SQL Reference*]
- "EXCEPT statement" [*SQL Anywhere Server - SQL Reference*]
- "INTERSECT statement" [*SQL Anywhere Server - SQL Reference*]
- "Supply explicit selectivity estimates sparingly" [*SQL Anywhere Server - SQL Usage*]
- "Explicit selectivity estimates" [*SQL Anywhere Server - SQL Reference*]
- "user_estimates connection property" on page 643

uuid_has_hyphens option

Controls the formatting of unique identifier values when they are converted to strings.

Allowed values

On, Off

Default

On

Remarks

When the uuid_has_hyphens option is set to On, the resulting strings contain four hyphens. For example, 12345678-1234-1234-1234-1234567890ab. With this option set to Off, the hyphens are omitted from the string. The database server supports UUID strings both with and without hyphens when converting a string to a unique identifier value.

See also

- "UNIQUEIDENTIFIER data type" [*SQL Anywhere Server - SQL Reference*]

verify_all_columns option [SQL Remote]

Controls whether messages that contain updates published by the local database are sent with all column values included.

Allowed values

On, Off

Default

Off

Remarks

This option is used by SQL Remote only. When it is set to On, messages that contain updates published by the local database are sent with all column values included, and a conflict in any column triggers a RESOLVE UPDATE trigger at the subscriber database.

Example

The following statement sets the verify_all_columns option to Off in SQL Anywhere, for all users:

```
SET OPTION PUBLIC.verify_all_columns = 'Off';
```

See also

- "SQL Remote options" [*SQL Remote*]
- "Using the verify_all_columns option" [*SQL Remote*]

verify_password_function option

Implements password rules.

Allowed values

String

Default

Empty string (no function is called when a password is set).

Scope

DBA authority required.

Remarks

The function specified by the verify_password_function is called automatically when a non-NULL password is created or set. To prevent a user from overriding the function, set the option value to *owner.function-name*. A user must have a password to be able to connect to the database. Passwords are case sensitive and they cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons
- be longer than 255 bytes in length

When passwords are created or changed, they are converted to UTF-8 before being hashed and stored in the database. If the database is unloaded and reloaded into a database with a different character set, existing passwords continue to work. If the database server cannot convert from the client's character set to UTF-8, then it is recommended that the password be composed of 7-bit ASCII characters as other characters may not work correctly.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can use any of the following statements to set a password:

- CREATE USER
- ALTER USER
- GRANT

After validating the statement used to create or set the password, the function is called to verify the password using the specified rules. If the password conforms to the specified rules, the function must return NULL to indicate success, and the invoking statement executes. Otherwise, an error is indicated by setting an error or returning a non-NULL string. If a non-NULL string is returned, it is included in the error to the user as the reason for failure.

The password verification function takes two parameters: *user-name* VARCHAR(128) and *new-pwd* VARCHAR(255). It returns a value of type VARCHAR(255). It is recommended that you execute an ALTER FUNCTION *function-name* SET HIDDEN statement on the password verification function to ensure that it cannot be stepped through using the debugger. If the verify_password_function option is set, specifying more than one user ID and password with a GRANT CONNECT statement is not allowed.

For more information about password rules, see "Use password verification" on page 1119.

See also

- "min_password_length option" on page 560
- "CREATE USER statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE FUNCTION statement" [*SQL Anywhere Server - SQL Reference*]
- "ALTER FUNCTION statement" [*SQL Anywhere Server - SQL Reference*]
- "ALTER USER statement" [*SQL Anywhere Server - SQL Reference*]
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "Increasing password security" on page 1118
- "NewPassword (NEWPWD) connection parameter" on page 299
- "verify_password_function connection property" on page 643

Example

The following example defines a table and a function and sets some login policy options. Together they implement advanced password rules that include requiring certain types of characters in the password, disallowing password reuse, and expiring passwords. The function is called by the database server with the verify_password_function option when a user ID is created or a password is changed. The application can call the procedure specified by the post_login_procedure option to report that the password should be changed before it expires.

The code for this sample is also available in the following location: *samples-dir\SQLAnywhere\SQL \verify_password.sql*. (For information about *samples-dir*, see "Samples directory" on page 392.)

```
-- only DBA should have permissions on this table
CREATE TABLE DBA.t_pwd_history(
        pk          INT         DEFAULT AUTOINCREMENT PRIMARY KEY,
        user_name   CHAR(128),  -- the user whose password is set
        pwd_hash    CHAR(32) ); -- hash of password value to detect
                                -- duplicate passwords

-- called whenever a non-NULL password is set
-- to verify the password conforms to password rules
```

```
CREATE FUNCTION DBA.f_verify_pwd( uid     VARCHAR(128),
                                  new_pwd VARCHAR(255) )
RETURNS VARCHAR(255)
BEGIN
    -- a table with one row per character in new_pwd
    DECLARE local temporary table pwd_chars(
            pos INT PRIMARY KEY,    -- index of c in new_pwd
            c   CHAR( 1 CHAR ) );   -- character
    -- new_pwd with non-alpha characters removed
    DECLARE pwd_alpha_only      CHAR(255);
    DECLARE num_lower_chars     INT;

    -- enforce minimum length (can also be done with
    -- min_password_length option)
    IF length( new_pwd ) < 6 THEN
        RETURN 'password must be at least 6 characters long';
    END IF;

    -- break new_pwd into one row per character
    INSERT INTO pwd_chars SELECT row_num, substr( new_pwd, row_num, 1 )
                            FROM dbo.RowGenerator
                            WHERE row_num <= length( new_pwd );

    -- copy of new_pwd containing alpha-only characters
    SELECT list( c, '' ORDER BY pos ) INTO pwd_alpha_only
        FROM pwd_chars WHERE c BETWEEN 'a' AND 'z' OR c BETWEEN 'A' AND 'Z';

    -- number of lower case characters IN new_pwd
    SELECT count(*) INTO num_lower_chars
        FROM pwd_chars WHERE CAST( c AS BINARY ) BETWEEN 'a' AND 'z';

    -- enforce rules based on characters contained in new_pwd
    IF ( SELECT count(*) FROM pwd_chars WHERE c BETWEEN '0' AND '9' )
           < 1 THEN
        RETURN 'password must contain at least one numeric digit';
    ELSEIF length( pwd_alpha_only ) < 2 THEN
        RETURN 'password must contain at least two letters';
    ELSEIF num_lower_chars = 0
           OR length( pwd_alpha_only ) - num_lower_chars = 0 THEN
        RETURN 'password must contain both upper- and lowercase characters';
    END IF;

    -- not the same as any user name
    -- (this could be modified to check against a disallowed words table)
    IF EXISTS( SELECT * FROM SYS.SYSUSER
                   WHERE lower( user_name ) IN ( lower( pwd_alpha_only ),
                                                 lower( new_pwd ) ) ) THEN
        RETURN 'password or only alphabetic characters in password ' ||
               'must not match any user name';
    END IF;

    -- not the same as any previous password for this user
    IF EXISTS( SELECT * FROM t_pwd_history
                   WHERE user_name = uid
                     AND pwd_hash = hash( uid || new_pwd, 'md5' ) ) THEN
        RETURN 'previous passwords cannot be reused';
    END IF;

    -- save the new password
    INSERT INTO t_pwd_history( user_name, pwd_hash )
        VALUES( uid, hash( uid || new_pwd, 'md5' ) );

    RETURN( NULL );
END;
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
ALTER FUNCTION DBA.f_verify_pwd SET HIDDEN;
GRANT EXECUTE ON DBA.f_verify_pwd TO PUBLIC;
SET OPTION PUBLIC.verify_password_function = 'DBA.f_verify_pwd';


-- All passwords expire in 180 days. Expired passwords can be changed
-- by the user using the NewPassword connection parameter.
ALTER LOGIN POLICY DEFAULT password_life_time = 180;

-- If an application calls the procedure specified by the
-- post_login_procedure option, then the procedure can be used to
-- warn the user that their password is about to expire. In particular,
-- Interactive SQL and Sybase Central call the post_login_procedure.
ALTER LOGIN POLICY DEFAULT password_grace_time = 30;

-- Five consecutive failed login attempts will result in a non-DBA
-- user ID being locked.
ALTER LOGIN POLICY DEFAULT max_failed_login_attempts = 5;
```

verify_threshold option [SQL Remote]

Controls which columns are verified when updates are replicated.

Allowed values

Integer, in bytes

Default

1000

Remarks

This option is used by SQL Remote only. If the data type of a column is longer than the threshold, old values for the column are not verified when an UPDATE is replicated. This keeps the size of SQL Remote messages down, but has the disadvantage that conflicting updates of long values are not detected.

See also

- "SQL Remote options" [*SQL Remote*]
- "BLOBs" [*SQL Remote*]

wait_for_commit option

Determines when foreign key integrity is checked, as data is manipulated.

Allowed values

On, Off

Default

Off

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Remarks

If this option is set to On, the database does not check foreign key integrity until the next COMMIT statement. Otherwise, all foreign keys that are not created with the check_on_commit option are checked as they are inserted, updated or deleted.

See also

- "sa_check_commit system procedure" [*SQL Anywhere Server - SQL Reference*]
- "COMMIT statement" [*SQL Anywhere Server - SQL Reference*]
- "Locking during inserts" [*SQL Anywhere Server - SQL Usage*]
- "Referential integrity checking" [*SQL Anywhere Server - SQL Usage*]
- "wait_for_commit connection property" on page 643

webservice_namespace_host option

Specifies the hostname to within the XML namespace specification for DISH services.

Allowed values

NULL or hostname-string

Default

NULL

Scope

Can be set for the PUBLIC group only. Takes effect immediately. DBA authority required.

Remarks

This option over-rides the default usage of the HTTP Host request header.

Webservices Description Language Documents (WSDLs) are exported by DISH services. These are XML documents that contain descriptions of the available SOAP services. The URL of the targetNameSpace and the soapAction operations within this XML document contain a hostname. When this option is set to NULL, the default value, the hostname is that of the computer on which the database server is running. If this option is set to a string value, the string is used as the hostname instead. This option is intended for use when developing web service client applications that will, when deployed, target a host other than the one used for development.

See also

- "Using SQL Anywhere as an HTTP web server" [*SQL Anywhere Server - Programming*]
- "webservice_namespace_host connection property" on page 644

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Connection, database, and database server properties

Connection properties

The following table lists properties available for each connection to a SQL Anywhere database.

You can use the CONNECTION_PROPERTY system function to retrieve the value for an individual property, or you can use the sa_conn_properties system procedure to retrieve the values of all connection properties. Property names are case insensitive.

Examples

To retrieve the value of a connection property

- Use the CONNECTION_PROPERTY system function. The following statement returns the number of pages that have been read from file by the current connection.

```
SELECT CONNECTION_PROPERTY ( 'DiskRead' );
```

To retrieve the values of all connection properties

- Use the sa_conn_properties system procedure:

```
CALL sa_conn_properties( );
```

  A separate row appears for each connection.

See also

- "CONNECTION_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]
- "sa_conn_activity system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Database server properties" on page 644
- "Database properties" on page 659

Descriptions

Property name	Description
allow_nulls_by_default	Returns a value indicating whether columns created without specifying either NULL or NOT NULL are allowed to contain NULL values. See "allow_nulls_by_default option" on page 505.
allow_read_client_file	Returns a value indicating whether the database server allows the reading of files on a client computer. See "allow_read_client_file option" on page 506.
allow_snapshot_isolation	Returns a value indicating whether snapshot isolation is enabled or disabled. See "allow_snapshot_isolation option" on page 506.

Property name	Description
allow_write_client_file	Returns a value indicating whether the database server allows the writing of files to a client computer. See "allow_write_client_file option" on page 508.
ansi_blanks	Returns a value indicating when character data is truncated at the client side. See "ansi_blanks option" on page 508.
ansi_close_cursors_on_rollback	Returns a value indicating whether cursors opened WITH HOLD are closed when a ROLLBACK is performed. See "ansi_close_cursors_on_rollback option" on page 509.
ansi_permissions	Returns a value indicating whether permissions are checked for DELETE and UPDATE statements. See "ansi_permissions option" on page 510.
ansi_substring	Returns a value indicating how the SUBSTRING (SUBSTR) function behaves when negative values are provided for the start or length parameters. See "ansi_substring option" on page 511.
ansi_update_constraints	Returns a value indicating the range of updates that are permitted. See "ansi_update_constraints option" on page 512.
ansinull	Returns a value that indicates how NULL values are interpreted. See "ansinull option" on page 513.
AppInfo	Returns information about the client that made the connection. For HTTP connections, this includes information about the browser. For connections using older versions of jConnect or Sybase Open Client, the information may be incomplete. The API value can be DBLIB, ODBC, OLEDB, ADO.NET, iAnywhereJDBC, PHP, PerlDBD, or DBEXPRESS. For more information about the values returned for other types of connections, see "AppInfo (APP) connection parameter" on page 266.
ApproximateCPUTime	Returns an estimate of the amount of CPU time accumulated by a given connection, in seconds. The value returned may differ from the actual value by as much as 50%, although typical variations are in the 5-10% range. On multiprocessor computers, each CPU (or hyperthread or core) accumulates time, so the sum of accumulated times for all connections may be greater than the elapsed time. This property is supported on Windows and Linux.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
auditing	Returns On if the PUBLIC.auditing option is set to On. Otherwise, returns Off. If the auditing option is set to On, and the conn_auditing option is set to Off, the auditing connection property still returns On, even though the current connection is not being audited. See "Controlling auditing" on page 1123 and "auditing option" on page 514.
auditing_options	This property is reserved for system use. Do not change the setting of this option.
Authenticated	Returns Yes if the application has sent a valid connection authentication string. Returns No if the application has not sent a valid connection authentication string.
AuthType	Returns the type of authentication used when connecting. The value returned is one of Standard, Integrated, Kerberos, or an empty string. An empty string is returned for internal connections and connections for HTTP services that use AUTHORIZATION OFF.
background_priority	This property is deprecated. Returns a value indicating how much impact the current connection has on the performance of other connections. See "background_priority option [deprecated]" on page 515.
BlockedOn	Returns zero if the current connection isn't blocked, or if it is blocked, the connection number on which the connection is blocked because of a locking conflict.
blocking	Returns a value indicating the database server's behavior in response to locking conflicts. See "blocking option" on page 516.
blocking_others_timeout	Returns the length of time that another connection can block on the current connection's row and table locks before the current connection is rolled back. See "blocking_others_timeout option" on page 517.
blocking_timeout	Returns the length of time, in milliseconds, a transaction waits to obtain a lock. See "blocking_timeout option" on page 517.
BytesReceived	Returns the number of bytes received during client/server communications. This value is updated for HTTP and HTTPS connections.
BytesReceivedUncomp	Returns the number of bytes that would have been received during client/server communications if compression was disabled. This value is the same as the value for BytesReceived if compression is disabled.
BytesSent	Returns the number of bytes sent during client/server communications. This value is updated for HTTP and HTTPS connections.

Property name	Description
BytesSentUncomp	Returns the number of bytes that would have been sent during client/server communications if compression was disabled. This value is the same as the value for BytesSent if compression is disabled.
CacheHits	Returns the number of successful reads of the cache.
CacheRead	Returns the number of database pages that have been looked up in the cache.
CacheReadIndInt	Returns the number of index internal-node pages that have been read from the cache.
CacheReadIndLeaf	Returns the number of index leaf pages that have been read from the cache.
CacheReadTable	Returns the number of table pages that have been read from the cache.
CacheReadWorkTable	Returns the number of cache work table reads .
CarverHeapPages	Returns the number of heap pages used for short-term purposes such as query optimization.
chained	Returns the transaction mode used in the absence of a BEGIN TRANSACTION statement. See “chained option” on page 518.
CharSet	Returns the CHAR character set used by the connection.
checkpoint_time	Returns the maximum time, in minutes, that the database server runs without doing a checkpoint. See “checkpoint_time option” on page 519.
cis_option	Returns 7 if debugging information for remote data access appears in the database server messages window and 0 if the debugging information for remote data access does not appear in the database server messages window. See “cis_option option” on page 519.
cis_rowset_size	Returns the number of rows that are returned from remote servers for each fetch. See “cis_rowset_size option” on page 520.
ClientLibrary	Returns jConnect for jConnect connections; CT_Library for Sybase Open Client connections; None for HTTP connections, and CmdSeq for ODBC, embedded SQL, OLE DB, ADO.NET, and SQL Anywhere JDBC driver connections.
ClientNodeAddress	Returns the node for the client in a client/server connection. When the client and server are both on the same computer, an empty string is returned. This is a synonym for the NodeAddress property. This property returns **NA** if the request that is currently executing is part of an event handler.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
ClientPort	Returns the client's TCP/IP port number or 0 if the connection isn't a TCP/IP connection.
ClientStmtCacheHits	Returns the number of prepares that were not required for this connection because of the client statement cache. This is the number of additional prepares that would be required if client statement caching was disabled. See "max_client_statements_cached option" on page 552.
ClientStmtCacheMisses	Returns the number of statements in the client statement cache for this connection that were prepared again. This is the number of times a cached statement was considered for reuse, but could not be reused because of a schema change, a database option setting, or a DROP VARIABLE statement. See "max_client_statements_cached option" on page 552.
close_on_endtrans	Returns On or Off to indicate whether cursors are closed at the end of a transaction. See "close_on_endtrans option" on page 520.
collect_statistics_on_dml_updates	Returns On or Off to indicate whether statistics are gathered during the execution of data-altering DML statements such as INSERT, DELETE, and UPDATE. See "collect_statistics_on_dml_updates option" on page 521.
Commit	Returns the number of Commit requests that have been handled.
CommLink	Returns the communication link for the connection. This is one of the network protocols supported by SQL Anywhere, or local for a same-computer connection. This property returns **NA** if the request that is currently executing is part of an event handler.
CommNetworkLink	Returns the communication link for the connection. This is one of the network protocols supported by SQL Anywhere. Values include SharedMemory and TCPIP. The CommNetworkLink property always returns the name of the link, regardless of whether it is same-computer or not. This property returns **NA** if the request that is currently executing is part of an event handler.
CommProtocol	Returns TDS for Sybase Open Client and jConnect connections, HTTP for HTTP connections, and CmdSeq for ODBC, embedded SQL, OLE DB, ADO.NET, and SQL Anywhere JDBC driver connections.

Property name	Description
Compression	Returns On or Off to indicate whether communication compression is enabled on the connection. This property returns **NA** if the request that is currently executing is part of an event handler.
conn_auditing	Returns On if auditing is enabled for the connection, even if the auditing option is set to Off. See "Controlling auditing" on page 1123 and "conn_auditing option" on page 522.
connection_authentication	Returns the string used to authenticate the client. Authentication is required before the database can be modified. See "connection_authentication option" on page 523.
continue_after_raiserror	Returns On or Off to indicate whether execution of a procedure or trigger is stopped whenever the RAISERROR statement is encountered. See "continue_after_raiserror option" on page 524.
conversion_error	Returns On or Off to indicate data type conversion failures are reported when fetching information from the database. See "conversion_error option" on page 525.
cooperative_commit_timeout	Returns the time, in milliseconds, that the database server waits for other connections to fill a page of the log before writing to disk. See "cooperative_commit_timeout option" on page 526.
cooperative_commits	Returns On or Off to indicate when commits are written to disk. See "cooperative_commits option" on page 526.
CurrentLineNumber	Returns the current line number of the procedure or compound statement a connection is executing. The procedure can be identified using the CurrentProcedure property. If the line is part of a compound statement from the client, an empty string is returned.
CurrentProcedure	Returns the name of the procedure that a connection is currently executing. If the connection is executing nested procedure calls, the name is the name of the current procedure. If there is no procedure executing, an empty string is returned.
Cursor	Returns the number of declared cursors that are currently being maintained by the server.
CursorOpen	Returns the number of open cursors that are currently being maintained by the server.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
database_authentication	Returns the string used to authenticate the database. Authentication is required for authenticated database servers before the database can be modified. See "database_authentication" on page 527.
date_format	Returns a string indicating the format for dates retrieved from the database. See "date_format option" on page 528.
date_order	Returns a string indicating how dates are formatted. See "date_order option" on page 530.
DBNumber	Returns the ID number of the database.
debug_messages	Returns On or Off to indicate whether MESSAGE statements that include a DEBUG ONLY clause are executed. See "debug_messages option" on page 531.
dedicated_task	Returns On or Off to indicate whether a request handling task is dedicated exclusively to handling requests for the connection. See "dedicated_task option" on page 531.
default_dbspace	Returns the name of the default dbspace, or an empty string if the default dbspace has not been specified. See "default_dbspace option" on page 532.
default_timestamp_increment	Returns a value, in microseconds, that is added to a column of type TIMESTAMP to keep values in the column unique. See "default_timestamp_increment option" on page 533.
delayed_commit_timeout	Returns the time, in milliseconds, that the database server waits to return control to an application following a COMMIT. See "delayed_commit_timeout option" on page 534.
delayed_commits	Returns On or Off to indicate when the database server returns control to an application following a COMMIT. See "delayed_commits option" on page 534.
DiskRead	Returns the number of pages that have been read from disk.
DiskReadHint	Returns the number of disk read hints.
DiskReadHintPages	Returns the number of disk read hint pages.
DiskReadIndInt	Returns the number of index internal-node pages that have been read from disk.
DiskReadIndLeaf	Returns the number of index leaf pages that have been read from disk.
DiskReadTable	Returns the number of table pages that have been read from disk.

Property name	Description
DiskReadWorkTable	Returns the number of disk work table reads.
DiskSyncRead	Returns the number of disk reads issued synchronously.
DiskSyncWrite	Returns the number of writes issued synchronously.
DiskWaitRead	Returns the number of times the database server waited for an asynchronous read.
DiskWaitWrite	Returns the number of times the database server waited for an asynchronous write.
DiskWrite	Returns the number of modified pages that have been written to disk.
DiskWriteHint	Returns the number of disk write hints.
DiskWriteHintPages	Returns the number of disk write hint pages.
divide_by_zero_error	Returns On if division by zero results in an error and Off if division by zero is not an error. See "divide_by_zero_error option" on page 536.
Encryption	Returns a value that indicates whether the connection is encrypted. See "Encryption (ENC) connection parameter" on page 287.
escape_character	This property is reserved for system use. Do not change the setting of this option.
EventName	Returns the name of the associated event if the connection is running an event handler. Otherwise, the result is NULL.
exclude_operators	This property is reserved for system use. Do not change the setting of this option.
ExprCacheAbandons	Returns the number of times that the expression cache was abandoned because the hit rate was too low.
ExprCacheDropsToReadOnly	Returns the number of times that the expression cache dropped to read-only status because the hit rate was low.
ExprCacheEvicts	Returns the number of evictions from the expression cache.
ExprCacheHits	Returns the number of hits in the expression cache.
ExprCacheInserts	Returns the number of values inserted into the expression cache.
ExprCacheLookups	Returns the number of lookups done in the expression cache.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
ExprCacheResumesOfReadWrite	Returns the number of times that the expression cache resumed read-write status because the hit rate increased.
ExprCacheStarts	Returns the number of times that the expression cache was started.
extended_join_syntax	Returns On if queries with duplicate correlation name syntax for multi-table joins are allowed, Off if they are reported as an error. See “extended_join_syntax option” on page 537.
fire_triggers	Returns On if triggers are fired in the database, otherwise, returns Off. See “fire_triggers option” on page 538.
first_day_of_week	Returns the number that is used for the first day of the week, where 7=Sunday and 1=Monday. See “first_day_of_week option” on page 538.
for_xml_null_treatment	Returns Omit if elements and attributes that contain NULL values are omitted from the result and Empty if empty elements or attributes are generated for NULL values when the FOR XML clause is used in a query. See “for_xml_null_treatment option” on page 539.
force_view_creation	This property is reserved for system use. Do not change the setting of this option.
FullCompare	Returns the number of comparisons that have been performed beyond the hash value in an index.
GetData	Returns the number of GETDATA requests.
global_database_id	Returns the starting value used for columns created with DEFAULT GLOBAL AUTOINCREMENT. See “global_database_id option” on page 540.
HashForcedPartitions	Returns the number of times that a hash operator was forced to partition because of competition for memory.
HashRowsFiltered	Returns the number of probe rows rejected by bit-vector filters.
HashRowsPartitioned	Returns the number of rows written to hash work tables.
HashWorkTables	Returns the number of work tables created for hash-based operations.
HeapsCarver	Returns the number of heaps used for short-term purposes such as query optimization..
HeapsLocked	Returns number of relocatable heaps currently locked in the cache.
HeapsQuery	Returns the number of heaps used for query processing (hash and sort operations).

Property name	Description
HeapsRelocatable	Returns the number of relocatable heaps.
http_connection_pool_basesize	Returns the nominal threshold size of database connections. See "http_connection_pool_basesize option" on page 541.
http_connection_pool_timeout	Returns the maximum length of time that unused connections are stored in the connection pool. See "http_connection_pool_timeout option" on page 542.
http_session_timeout	Returns the current HTTP session timeout, in minutes. See "http_session_timeout option" on page 542.
HttpServiceName	Returns the service name entry point for the current HTTP request. The property is useful for error reporting and control flow. An empty string is returned when this property is selected from a stored procedure that did not originate from an HTTP request or if the connection is currently inactive waiting to continue an HTTP session.
IdleTimeout	Returns the idle timeout value of the connection. See "Idle connection parameter" on page 292. This property returns **NA** if the request that is currently executing is part of an event handler.
IndAdd	Returns the number of entries that have been added to indexes.
IndLookup	Returns the number of entries that have been looked up in indexes.
integrated_server_name	Returns the name of the Domain Controller server used for looking up Windows user group membership for integrated logins. See "integrated_server_name option" on page 543.
IsDebugger	Returns Yes or No to distinguish connections that are being used to run the SQL Anywhere debugger. The value is Yes if the current connection number corresponds to the connection number of a debugger connection, and No otherwise. See "Debugging procedures, functions, triggers, and events" [*SQL Anywhere Server - SQL Usage*].
isolation_level	Returns the isolation level of the connection (0, 1, 2, 3, Snapshot, Statement-snapshot, or Readonly-statement-snapshot). See "isolation_level option" on page 544.
java_location	Returns the path of the Java VM for the database if one has been specified. See "java_location option" on page 545.
java_main_userid	This property is deprecated.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
java_vm_options	Returns the command line options that the database server uses when it launches the Java VM. See "java_vm_options option" on page 546.
Language	Returns the locale language.
LastIdle	Returns the number of ticks between requests.
LastPlanText	Returns the long text plan of the last query executed on the connection. You control the remembering of the last plan by setting using the RememberLastPlan option of the sa_server_option system procedure, or using the -zp server option. See "-zp dbeng12/dbsrv12 server option" on page 247.
LastReqTime	Returns the time at which the last request for the specified connection started. This property can return an empty string for internal connections, such as events.
LastStatement	Returns the most recently prepared SQL statement for the current connection. See "-zl dbeng12/dbsrv12 server option" on page 244. The LastStatement value is set when a statement is prepared, and is cleared when a statement is dropped. Only one statement string is remembered for each connection. If sa_conn_activity reports a non-empty value for a connection, it is most likely the statement that the connection is currently executing. If the statement had completed, it would likely have been dropped and the property value would have been cleared. If an application prepares multiple statements and retains their statement handles, the LastStatement value does not reflect what a connection is currently doing. When client statement caching is enabled, and a cached statement is reused, this property returns an empty string.
LivenessTimeout	Returns the liveness timeout period for the current connection. See "LivenessTimeout (LTO) connection parameter" on page 297. This property returns **NA** if the request that is currently executing is part of an event handler.
lock_rejected_rows	This property is reserved for system use. Do not change the setting of this option.
LockCount	Returns the number of locks held by the connection.
LockIndexID	Returns the identifier of the locked index.
LockName	Returns a 64-bit unsigned integer value representing the lock for which a connection is waiting.

Property name	Description
LockRowID	Returns the identifier of the locked row.
LockTableOID	Returns zero if the connection isn't blocked, or if the connection is on a different database than the connection calling CONNECTION_PROPERTY. Otherwise, this is the object ID of the table for the lock on which this connection is waiting. The object ID can be used to look up table information using the SYSTAB system view. See "SYSTAB system view" [*SQL Anywhere Server - SQL Reference*].
log_deadlocks	Returns On if deadlock information is reported; otherwise, returns Off. See "log_deadlocks option" on page 547.
LogFreeCommit	Returns the number of redo free commits. A redo free commit occurs when a commit of the transaction log is requested but the log has already been written (so the commit was done for free.)
login_mode	Returns one or more of Standard, Integrated, or Kerberos to indicate whether integrated logins and Kerberos are supported. See "login_mode option" on page 547.
login_procedure	Returns the name of the stored procedure used to set compatibility options at startup. See "login_procedure option" on page 549.
LoginTime	Returns the date and time the connection was established.
LogWrite	Returns the number of pages that have been written to the transaction log.
materialized_view_optimization	Returns a value indicating whether materialized views are used during query optimization: • Disabled • Fresh • Stale • *N* Minute[s] • *N* Hour[s] • *N* Day[s] • *N* Week[s] • *N* Month[s] See "materialized_view_optimization option" on page 551.
max_client_statements_cached	Returns the number of statements cached by the client. See "max_client_statements_cached option" on page 552.
max_cursor_count	Returns a value specifying the maximum number of cursors that a connection can use at once. See "max_cursor_count option" on page 554.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
max_hash_size	This property is deprecated.
max_plans_cached	Returns a value specifying the maximum number of execution plans to be stored in a cache. See "max_plans_cached option" on page 554.
max_priority	Returns a value indicating the maximum priority level a connection can have. See "max_priority option" on page 555.
max_query_tasks	Returns the maximum number of requests that the database server can use to process a query. See "max_query_tasks option" on page 556.
max_recursive_iterations	Returns a value specifying the maximum number of iterations a recursive common table expression can make. See "max_recursive_iterations option" on page 557.
max_statement_count	Returns a value specifying the maximum number of prepared statements that a connection can use simultaneously. See "max_statement_count option" on page 557.
max_temp_space	Returns a value indicating the maximum amount of temporary file space available for a connection. See "max_temp_space option" on page 559.
MessageReceived	Returns the string that was generated by the MESSAGE statement that caused the WAITFOR statement to be interrupted. Otherwise, an empty string is returned.
min_password_length	Returns the minimum length for new passwords in the database. See "min_password_length option" on page 560.

Property name	Description
Name	Returns the name of the current connection. You can specify a connection name using the ConnectionName (CON) connection parameter. See "ConnectionName (CON) connection parameter" on page 278. The following names are used for temporary connections created by the database server: • BackupDB • Checkpoint • Cleaner • CloseDB • CreateDB • DelayedCommit • DiagRcvr • DropDB • EncryptDB • Exchange • FlushMirrorLog • FlushStats • HTTPReq • PurgeSnapshot • RecoverMirror • RedoCheckpoint • RefreshIndex • ReloadTrigger • RenameMirror • RestoreDB • StartDB • VSS For information about temporary connections, see "Temporary connections" on page 137.
NcharCharSet	Returns the NCHAR character set used by the connection.
nearest_century	Returns a value that indicates how two-digit years are interpreted in string-to-date conversions. See "nearest_century option" on page 561.
NodeAddress	Returns the node for the client in a client/server connection. When the client and server are both on the same computer, an empty string is returned.
non_keywords	Returns a list of keywords, if any, that are turned off so they can be used as identifiers. See "non_keywords option" on page 561.
Number	Returns the ID number of the connection.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
odbc_describe_binary_as_varbinary	Returns Off if the SQL Anywhere ODBC driver describes both BINARY and VARBINARY columns as SQL_BINARY and returns On if the ODBC driver describes BINARY and VARBINARY columns as SQL_VARBINARY. See "odbc_describe_binary_as_varbinary" on page 562.
odbc_distinguish_char_and_varchar	Returns Off if CHAR columns are described as SQL_VARCHAR, and On if CHAR columns are described as SQL_CHAR. See "odbc_distinguish_char_and_varchar option" on page 563.
oem_string	Returns the string stored in the header page of the database file. See "oem_string option" on page 563.
on_charset_conversion_failure	Returns one of Ignore, Warning, or Error to indicate the behavior when an error is encountered during character set conversion. See "on_charset_conversion_failure option" on page 565.
on_tsql_error	Returns one of Stop, Conditional, or Continue to indicate the behavior when an error is encountered while executing a stored procedure. See "on_tsql_error option" on page 566.
optimization_goal	Returns one of First-row or All-rows to indicate how query processing is optimized. See "optimization_goal option" on page 567.
optimization_level	Returns a value between 0 and 15. This number is used to control the amount of effort made by the SQL Anywhere query optimizer to find an access plan for a SQL statement. See "optimization_level option" on page 568.
optimization_workload	Returns a value indicating the amount of effort made by the SQL Anywhere query optimizer to find an access plan for a SQL statement. See "optimization_workload option" on page 569.
OSUser	Returns the operating system user name associated with the client process. If the client process is impersonating another user (or the set ID bit is set on Unix), the impersonated user name is returned. An empty string is returned for version 10.0.1 and earlier clients, and for HTTP and TDS clients.
PacketSize	Returns the packet size used by the connection, in bytes. This property returns **NA** if the request that is currently executing is part of an event handler.
PacketsReceived	Returns the number of client/server communication packets received. This value is not updated for HTTP or HTTPS connections.

Property name	Description
PacketsReceivedUncomp	Returns the number of packets that would have been received during client/server communications if compression was disabled. (This value is the same as the value for PacketsReceived if compression is disabled.)
PacketsSent	Returns the number of client/server communication packets sent. This value is not updated for HTTP or HTTPS connections.
PacketsSentUncomp	Returns the number of packets that would have been sent during client/server communications if compression was disabled. (This value is the same as the value for PacketsSent if compression is disabled.)
ParentConnection	Returns the connection ID of the connection that created a temporary connection to perform a database operation (such as a performing a backup or creating a database). For other types of connections, this property returns NULL. See "Temporary connections" on page 137.
pinned_cursor_percent_of_cache	Returns the percentage of the cache that can be used for pinning cursors. See "pinned_cursor_percent_of_cache option" on page 570.
post_login_procedure	Returns the name of the procedure whose result set contains messages that should be displayed by applications when a user connects. See "post_login_procedure option" on page 571.
precision	Returns the decimal and numeric precision setting. See "precision option" on page 573.
prefetch	Returns Off if no prefetching is done, Conditional if prefetching occurs unless the cursor type is SENSITIVE or the query includes a proxy table, or Always if prefetching is done even for SENSITIVE cursor types and cursors that involve a proxy table. See "prefetch option" on page 573.
Prepares	Returns the number of statement preparations performed for the connection.
PrepStmt	Returns the number of prepared statements currently being maintained by the server.
preserve_source_format	Returns On if the original source definition of procedures, triggers, views, and event handlers is saved in system tables, otherwise, returns Off. See "preserve_source_format option" on page 574.
prevent_article_pkey_update	Returns On if updates are not allowed to the primary key columns of tables involved in publications, otherwise returns Off. See "prevent_article_pkey_update option" on page 575.
priority	Returns a value indicating the priority level of a connection. See "priority option" on page 575.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
Progress	Returns information about how long a query has been running. For example: `43% (9728 of 22230 pages) complete after 00:00:05; estimated 00:00:06 remaining`
progress_messages	Returns the value of the progress_messages option. See "progress_messages option" on page 576.
query_mem_timeout	Returns the value of the query_mem_timeout option. See "query_mem_timeout option" on page 578.
QueryBypassed	Returns the number of requests optimized by the optimizer bypass.
QueryBypassedCosted	Returns the number of requests processed by the optimizer bypass using costing.
QueryBypassedHeuristic	Returns the number of requests processed by the optimizer bypass using heuristics.
QueryBypassedOptimized	Returns the number of requests initially processed by the optimizer bypass and subsequently fully optimized by the SQL Anywhere optimizer.
QueryCachedPlans	Returns the number of query execution plans currently cached for the connection.
QueryCachePages	Returns the number of cache pages used to cache execution plans.
QueryDescribedBypass	Returns the number of describe requests processed by the optimizer bypass.
QueryDescribedOptimizer	Returns the number of describe requests processed by the optimizer.
QueryHeapPages	Returns the number of cache pages used for query processing (hash and sort operations).
QueryJHToJNLOptUsed	Returns the number of times a hash join was converted to a nested loops join.
QueryLowMemoryStrategy	Returns the number of times the server changed its execution plan during execution as a result of low memory conditions. The strategy can change because less memory is now available than the optimizer estimated, or because the execution plan required more memory than the optimizer estimated.
QueryMemActiveCurr	Returns the number of requests actively using query memory.

Property name	Description
QueryMemGrant-Failed	Returns the total number of times a request waited for query memory, but failed to get it.
QueryMemGrant-Granted	Returns the number of pages currently granted to requests.
QueryMemGran-tRequested	Returns the total number of times any request attempted to acquire query memory.
QueryMemGrant-Waited	Returns the total number of times any request waited for query memory.
QueryMemGrant-Waiting	Returns the current number of requests waiting for query memory.
QueryOpened	Returns the number of OPEN requests for execution.
QueryOptimized	Returns the number of requests fully optimized.
QueryReused	Returns the number of requests that have been reused from the plan cache.
QueryRowsFetched	Returns the number of rows that have been read from base tables, either by a sequential scan or an index scan, for this connection.
QueryRowsMateri-alized	Returns the number of rows are written to work tables during query processing.
quoted_identifier	Returns On if strings enclosed in double quotes are interpreted as identifiers, or Off if they are interpreted as literal strings. See "quoted_identifier option" on page 579.
read_past_deleted	Returns On if sequential scans at isolation levels 1 and 2 skip uncommitted deleted rows, and Off if sequential scans block on uncommitted deleted rows at isolation levels 1 and 2. See "read_past_deleted option" on page 580.
recovery_time	Returns the maximum length of time, in minutes, that the database server will take to recover from system failure. See "recovery_time option" on page 580.
RecursiveIterations	Returns the number of iterations for recursive unions.
RecursiveIteration-sHash	Returns the number of times recursive hash join used a hash strategy.
RecursiveIterations-Nested	Returns the number of times recursive hash join used a nested loops strategy.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
RecursiveJNLMisses	Returns the number of index probe cache misses for recursive hash join.
RecursiveJNLProbes	Returns the number of times recursive hash join attempted an index probe.
remote_idle_timeout	Returns the time, in seconds, of inactivity that web service client procedures and functions will tolerate. See "remote_idle_timeout option" on page 581.
replicate_all	For internal use only.
ReqCountActive	Returns the number of requests processed, or NULL if the RequestTiming server property is set to Off. See "-zt dbeng12/dbsrv12 server option" on page 250.
ReqCountBlockContention	Returns the number of times the connection waited for atomic access, or NULL if the -zt option was not specified. See "-zt dbeng12/dbsrv12 server option" on page 250.
ReqCountBlockIO	Returns the number of times the connection waited for I/O to complete, or NULL if the -zt option was not specified. See "-zt dbeng12/dbsrv12 server option" on page 250.
ReqCountBlockLock	Returns the number of times the connection waited for a lock, or NULL if the -zt option was not specified. See "-zt dbeng12/dbsrv12 server option" on page 250.
ReqCountUnscheduled	Returns the number of times the connection waited for scheduling, or NULL if the -zt option was not specified. See "-zt dbeng12/dbsrv12 server option" on page 250.

Property name	Description
ReqStatus	Returns the status of the request. It can be one of the following values: • **Idle** The connection is not currently processing a request. • **Unscheduled*** The connection has work to do and is waiting for an available database server worker. • **BlockedIO*** The connection is blocked waiting for an I/O. • **BlockedContention*** The connection is blocked waiting for access to shared database server data structures. • **BlockedLock** The connection is blocked waiting for a locked object. • **Executing** The connection is executing a request. The values marked with an asterisk (*) are only returned when logging of request timing information has been turned on for the database server using the -zt server option. If request timing information is not being logged (the default), the values are reported as Executing. For more information, see "-zt dbeng12/dbsrv12 server option" on page 250.
ReqTimeActive	Returns the amount of time spent processing requests, or NULL if the -zt option was not specified. See "-zt dbeng12/dbsrv12 server option" on page 250.
ReqTimeBlockContention	Returns the amount of time spent waiting for atomic access, or NULL if the RequestTiming server property is set to Off. See "-zt dbeng12/dbsrv12 server option" on page 250.
ReqTimeBlockIO	Returns the amount of time spent waiting for I/O to complete, or NULL if the -zt option was not specified. See "-zt dbeng12/dbsrv12 server option" on page 250.
ReqTimeBlockLock	Returns the amount of time spent waiting for a lock, or NULL if the -zt option was not specified. See "-zt dbeng12/dbsrv12 server option" on page 250.
ReqTimeUnscheduled	Returns the amount of unscheduled time, or NULL if the -zt option was not specified. See "-zt dbeng12/dbsrv12 server option" on page 250.
ReqType	Returns the type of the last request. The connection property for cached connections is ConnPoolCachedCount.
request_timeout	Returns the maximum time a single request can run. See "request_timeout option" on page 582.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
RequestsReceived	Returns the number of client/server communication requests or round trips. It is different from PacketsReceived in that multi-packet requests count as one request, and liveness packets are not included.
reserved_keywords	Returns a list of non-default reserved keywords that are enabled for the database. See "reserved_keywords option" on page 583.
return_date_time_as_string	Returns On if date, time, and timestamp values are returned to applications as a string, and Off if they are returned as a date or time data type. See "return_date_time_as_string option" on page 584.
Rlbk	The number of rollback requests that have been handled.
rollback_on_deadlock	Returns After when referential integrity actions are executed after the UPDATE or DELETE, and Before if they are executed before the UPDATE or DELETE. See "rollback_on_deadlock" on page 585.
RollbackLogPages	Returns the number of pages in the rollback log.
row_counts	Returns On if the row count is always accurate, and Off if the row count is usually an estimate. See "row_counts option" on page 585.
scale	Returns the decimal and numeric scale for the connection. See "scale option" on page 586.
secure_feature_key	Stores the key that is used to enable and disable features for a database server. Selecting the value of this property always returns an empty string.
ServerNodeAddress	Returns the node for the server in a client/server connection. When the client and server are both on the same computer, an empty string is returned. This property returns **NA** if the request that is currently executing is part of an event handler.
ServerPort	Returns the database server's TCP/IP port number or 0.
SessionCreateTime	Returns the time the HTTP session was created.
SessionID	Returns the session ID for the connection if it exists, otherwise, returns an empty string.
SessionLastTime	Returns the time of the last request for the HTTP session.
SessionTimeout	Returns the time, in minutes, the HTTP session persists during inactivity. See "sa_set_http_option system procedure" [*SQL Anywhere Server - SQL Reference*].

Property name	Description
SnapshotCount	Returns the number of snapshots associated with the connection.
sort_collation	Returns Internal if the ORDER BY clause remains unchanged, otherwise the collation name or collation ID is returned. See "sort_collation option" on page 588.
SortMergePasses	Returns the number of merge passes used during sorting.
SortRowsMaterialized	Returns the number of rows written to sort work tables.
SortRunsWritten	Returns the number of sorted runs written during sorting.
SortSortedRuns	Returns the number of sorted runs created during run formation.
SortWorkTables	Returns the number of work tables created for sorting.
sql_flagger_error_level	Returns one of the following values to indicate which SQL that is not part of a specified set of SQL/2003 is flagged as an error: • **E** Flag syntax that is not entry-level SQL/2003 syntax • **I** Flag syntax that is not intermediate-level SQL/2003 syntax • **F** Flag syntax that is not full-SQL/2003 syntax • **W** Allow all supported syntax For more information, see "sql_flagger_error_level option" on page 589.
sql_flagger_warning_level	Returns one of the following values to indicate which SQL that is not part of a specified set of SQL/2003 is flagged as a warning: • **E** Flag syntax that is not entry-level SQL/2003 syntax • **I** Flag syntax that is not intermediate-level SQL/2003 syntax • **F** Flag syntax that is not full-SQL/2003 syntax • **W** Allow all supported syntax For more information, see "sql_flagger_warning_level option" on page 590.
st_geometry_asbinary_format	Returns a value that indicates how spatial values are converted from a geometry to binary. See "st_geometry_asbinary_format option" on page 594.
st_geometry_astext_format	Returns a value that indicates how spatial values are converted from a geometry to text. See "st_geometry_astext_format option" on page 595.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
st_geometry_asxml_format	Returns a value that indicates how spatial values are converted from a geometry to xml. See "st_geometry_asxml_format option" on page 596.
st_geometry_describe_type	Returns a value that indicates spatial values are described. See "st_geometry_describe_type option" on page 597.
st_geometry_on_invalid	Returns a value that indicates the behavior when a geometry fails surface validation. See "st_geometry_on_invalid option" on page 598.
StatementDescribes	Returns the total number of statements processed by DESCRIBE requests.
StatementPostAnnotates	Returns the number of statements processed by the semantic query transformation phase.
StatementPostAnnotatesSimple	Returns the number of statements processed by the semantic query transformation phase, but which skipped some of the semantic transformations.
StatementPostAnnotatesSkipped	Returns the number of statements that have completely skipped the semantic query transformation phase.
string_rtruncation	Returns On if an error is raised when a string is truncated, and returns Off if an error is not raised and the character string is silently truncated. See "string_rtruncation option" on page 598.
subsume_row_locks	Returns On if the database server acquires individual row locks for a table, otherwise, returns Off. See "subsume_row_locks option" on page 600.
suppress_tds_debugging	Returns Off if TDS debugging information appears in the database server messages window, and returns On if debugging information does not appear in the database server messages window. See "suppress_tds_debugging option" on page 600.
synchronize_mirror_on_commit	Returns On if the database mirror server is synchronized on commit, otherwise returns Off. See "synchronize_mirror_on_commit option" on page 601.
tds_empty_string_is_null	Returns On if empty strings are returned as NULL for TDS connections, and returns Off if a string containing one blank character is returned for TDS connections. See "tds_empty_string_is_null option" on page 601.
temp_space_limit_check	Returns On if the database server checks the amount of temporary space available for a connection, and returns Off if the database server does not check the amount of space available for a connection. See "temp_space_limit_check option" on page 602.
TempFilePages	Returns the number of temporary file pages used by the connection
TempTablePages	Returns the number of pages in the temporary file used for temporary tables.

Property name	Description
time_format	Returns the string format used for times retrieved from the database. See "time_format option" on page 603.
time_zone_adjustment	Returns the number of minutes that must be added to the Coordinated Universal Time (UTC) to display time local to the connection. See "time_zone_adjustment option" on page 604.
timestamp_format	Returns the number of minutes that must be added to the Coordinated Universal Time (UTC) to display time local to the connection. See "timestamp_format option" on page 604.
timestamp_with_time_zone_format	Returns the format for TIMESTAMP WITH TIME ZONE values retrieved from the database. See "timestamp_with_time_zone_format option" on page 606.
TimeZoneAdjustment	Returns the number of minutes that must be added to the Coordinated Universal Time (UTC) to display time local to the connection. See "time_zone_adjustment option" on page 604.
TransactionStartTime	Returns a string containing the time the database was first modified after a COMMIT or ROLLBACK, or an empty string if no modifications have been made to the database since the last COMMIT or ROLLBACK.
truncate_timestamp_values	Returns On if the number of decimal places used in the timestamp values is limited, otherwise, returns Off. See "truncate_timestamp_values option" on page 608.
tsql_outer_joins	Returns On if Transact-SQL outer joins can be used in DML statements. See "tsql_outer_joins option" on page 609.
tsql_variables	Returns On if you can use the @ sign instead of the colon as a prefix for host variable names in embedded SQL, otherwise, returns Off. See "tsql_variables option" on page 610.
UncommitOp	Returns the number of uncommitted operations.
updatable_statement_isolation	Returns the isolation level (0, 1, 2, or 3) used by updatable statements when the isolation_level option is set to Readonly-statement-snapshot. See "updatable_statement_isolation option" on page 610.
update_statistics	Returns On if this connection can send query feedback to the statistics governor. When the update_statistics option is set to Off, the statistics governor does not receive query feedback from the current connection. See "update_statistics option" on page 611.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
upgrade_database_capability	This property is reserved for system use. Do not change the setting of this option.
user_estimates	Returns one of the following values that controls whether selectivity estimates in query predicates are respected or ignored by the query optimizer: • **Enabled** All user-supplied selectivity estimates are respected. You can also use On to turn on this option. • **Override-Magic** A user selectivity estimate is respected and used only if the optimizer would otherwise choose to use its last-resort, heuristic value (also called the magic value). • **Disabled** All user estimates are ignored and magic values are used when no other estimate data is available. You can also use Off to turn off this option. For more information, see "user_estimates option" on page 612.
UserAppInfo	Returns the string specified by the AppInfo connection parameter in a connection string. For more information, see "AppInfo (APP) connection parameter" on page 266.
UserID	Returns the user ID for the connection.
UtilCmdsPermitted	Returns On or Off to indicate whether utility commands such as CREATE DATABASE, DROP DATABASE, and RESTORE DATABASE are permitted for the connection. See "-gu dbeng12/dbsrv12 server option" on page 198.
verify_password_function	Returns the name of the function used for password verification if one has been specified. See "verify_password_function option" on page 614.
wait_for_commit	Returns On if the database does not check foreign key integrity until the next COMMIT statement. Otherwise, returns Off and all foreign keys that are not created with the check_on_commit option are checked as they are inserted, updated or deleted. See "wait_for_commit option" on page 617.
WaitStartTime	Returns the time at which the connection started waiting (or an empty string if the connection is not waiting).

Property name	Description
WaitType	Returns the reason for the wait, if it is available. Possible values for WaitType are: • **lock** Returned if the connection is waiting on a lock. • **waitfor** Returned if the connection is executing a waitfor statement. • **empty string** Returned if the connection is not waiting, or if the reason for the wait is not available.
webservice_name-space_host	Returns the hostname to be used as the XML namespace within generated WSDL documents if one has been specified. See "webservice_name-space_host option" on page 618.

Database server properties

The following table lists properties that apply across the database server as a whole.

You can use the PROPERTY system function to retrieve the value for an individual property, or you can use the sa_eng_properties system procedure to retrieve the values of all database server properties. Property names are case insensitive.

Examples

To retrieve the value of a database server property

- Use the PROPERTY system function. For example, the following statement returns the number of cache pages used for global server data structures:

```
SELECT PROPERTY ( 'MainHeapPages' );
```

To retrieve the values of all database server properties

- Use the sa_eng_properties system procedure:

```
CALL sa_eng_properties;
```

See also

- "PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]
- "Connection properties" on page 619
- "Database properties" on page 659

Descriptions

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
ActiveReq	Returns the number of server workers that are currently handling client-side requests.
AutoMultiProgrammingLevel	Returns a value indicating whether the database server is automatically adjusting its multiprogramming level. See "-gna dbsrv12 server option" on page 190.
AutoMultiProgrammingLevelStatistics	Returns a value indicating whether messages about automatic adjustments to the database server's multiprogramming level are displayed in the database server message log. See "-gns dbsrv12 server option" on page 192.
AvailIO	Returns the current number of available I/O control blocks.
BuildChange	This property is reserved for system use. Do not change the setting of this property.
BuildClient	This property is reserved for system use. Do not change the setting of this property.
BuildProduction	Returns Yes if the database server is compiled for production use or returns No if the database server is a debug build.
BuildReproducible	This property is reserved for system use. Do not change the setting of this property.
BytesReceived	Returns the number of bytes received during client/server communications. This value is updated for HTTP and HTTPS connections.
BytesReceivedUncomp	Returns the number of bytes that would have been received during client/server communications if compression was disabled. (This value is the same as the value for BytesReceived if compression is disabled.)
BytesSent	Returns the number of bytes sent during client/server communications. This value is updated for HTTP and HTTPS connections.
BytesSentUncomp	Returns the number of bytes that would have been sent during client/server communications if compression was disabled. (This value is the same as the value for BytesSent if compression is disabled.)
CacheAllocated	Returns the number of cache pages that have been allocated for server data structures.
CacheFile	Returns the number of cache pages used to hold data from database files.
CacheFileDirty	Returns the number of cache pages that are dirty (needing a write).
CacheFree	Returns the number of cache pages not being used.
CacheHits	Returns the number of database page lookups.

Property name	Description
CachePanics	Returns the number of times the cache manager has failed to find a page to allocate.
CachePinned	Returns the number of pinned cache pages.
CacheRead	Returns the number of cache reads.
CacheReplacements	Returns the number of pages in the cache that have been replaced.
CacheScavenges	Returns the number of times the cache manager has scavenged for a page to allocate.
CacheScavengeVisited	Returns the number of pages visited while scavenging for a page to allocate.
CacheSizingStatistics	Returns Yes if the server is displaying cache sizing statistics when the cache is resized, otherwise, returns No. See "-cs dbeng12/dbsrv12 server option" on page 169.
CarverHeapPages	Returns the number of heap pages used for short-term purposes such as query optimization.
CharSet	Returns the CHAR character set in use by the database server.
ClientStmtCacheHits	Returns the number of prepares that were not required because of the client statement cache. This is the number of additional prepares that would be required if client statement caching was disabled. See "max_client_statements_cached option" on page 552.
ClientStmtCacheMisses	Returns the number of statements in the client statement cache that were prepared again. This is the number of times a cached statement was considered for reuse, but could not be reused because of a schema change, a database option setting, or a DROP VARIABLE statement. See "max_client_statements_cached option" on page 552.
CollectStatistics	Returns Yes or No to indicate whether the database server is collecting performance statistics. See "-k dbeng12/dbsrv12 server option" on page 200.
CommandLine	Returns the command line that was used to start the database server. If the encryption key for a database was specified using the -ek option, the key is replaced with a constant string of asterisks in the value returned by this property.
CompactPlatformVer	Returns a condensed version of the PlatformVer property.
CompanyName	Returns the name of the company owning this software.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
ConnsDisabled	Returns Yes or No to indicate the current setting of the server option to disable new connections. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].
ConsoleLogFile	Returns the name of the file where database server messages are logged if the -o option was specified, otherwise returns an empty string. See "-o dbeng12/dbsrv12 server option" on page 208 and "Logging database server actions" on page 41.
ConsoleLogMaxSize	Returns the maximum size in bytes of the file used to log database server messages. See "-os dbeng12/dbsrv12 server option" on page 210.
CurrentCacheSize	Returns the current cache size, in kilobytes.
CurrentMultiProgrammingLevel	Returns the current number of tasks that the database server can process concurrently. See "Configuring the database server's multiprogramming level" on page 52.
DebuggingInformation	Returns Yes if the server is displaying diagnostic messages for troubleshooting, and No otherwise. See "-z dbeng12/dbsrv12 server option" on page 243.
DefaultCollation	Returns the collation that would be used for new databases if none is explicitly specified.
DefaultNcharCollation	Returns the name of the default NCHAR collation on the server computer (UCA if ICU is installed, and UTF8BIN otherwise).
DiskRead	Returns the number of disk reads.
DiskReadHintScatterLimit	Returns the imposed limit on the size (in bytes) of a scatter read hint.
DiskRetryRead	Returns the number of disk read retries.
DiskRetryReadScatter	Returns the number of disk read retries for scattered reads.
DiskRetryWrite	Returns the number of disk write retries.
EventTypeDesc	Returns the system event type description associated with a given event type ID.
EventTypeName	Returns the system event type name associated with a given event type ID.

Property name	Description
Exchange-Tasks	Returns the number of tasks currently being used for parallel execution of queries.
Exchange-TasksCompleted	Returns the total number of internal tasks that have been used for intra-query parallelism since the database server started. See "Parallelism during query execution" [*SQL Anywhere Server - SQL Usage*].
FipsMode	Returns Yes if the -fips option was specified when the database server was started, and No otherwise.
FirstOption	Returns the number that represents the first connection property that corresponds to a database option.
FreeBuffers	Returns the number of available network buffers.
FunctionMaxParms	Returns the maximum number of parameters that can be specified by a function. The function is identified by the value specified by the *function-number*, which is a positive integer. For example: `SELECT PROPERTY ( 'FunctionMaxParms', function-number );` Note that the *function-number* is subject to change between releases.
FunctionMinParms	Returns the minimum number of parameters that must be specified by a function. The function is identified by the value specified by the *function-number*, which is a positive integer. For example: `SELECT PROPERTY ( 'FunctionMaxParms', function-number );` Note that the *function-number* is subject to change between releases.
FunctionName	Returns the name of the function identified by the value specified by the *function-number* (which is a positive integer): `SELECT PROPERTY ( 'FunctionName', function-number );` Note that the *function-number* is subject to change between releases.
HeapsCarver	Returns the number of heaps used for short-term purposes such as query optimization.
HeapsLocked	Returns number of relocatable heaps currently locked in the cache.
HeapsQuery	Returns the number of heaps used for query processing (hash and sort operations).
HeapsRelocatable	Returns the number of relocatable heaps.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
HttpAddresses	Returns a semicolon delimited list of the TCP/IP addresses the server is listening to for HTTP connections. For example: `(::1):80;127.0.0.1:80`
HttpNumActiveReq	Returns the number of HTTP connections that are actively processing an HTTP request. An HTTP connection that has sent its response is not included.
HttpNumConnections	Returns the number of HTTP connections that are currently open within the database server. They may be actively processing a request or waiting in a queue of long lived (keep-alive) connections.
HttpNumSessions	Returns the number of active and dormant HTTP sessions within the database server.
HttpPorts	Returns the HTTP port numbers for the web server as a comma delimited list.
HttpsAddresses	Returns a semicolon delimited list of the TCP/IP addresses the server is listening to for HTTPS connections. For example: `(::1):443;127.0.0.1:443`
HttpsNumActiveReq	Returns the number of secure HTTPS connections that are actively processing an HTTPS request. An HTTPS connection that has sent its response is not included.
HttpsNumConnections	Returns the number of HTTPS connections that are currently open within the database server. They may be actively processing a request or waiting in a queue of long lived (keep-alive) connections.
HttpsPorts	Returns the HTTPS port numbers for the web server as a comma delimited list.
IdleTimeout	Returns the default idle timeout. See "-ti dbeng12/dbsrv12 server option" on page 226.
IPAddressMonitorPeriod	Returns the time in seconds that a database server checks for new IP addresses. See "-xm dbeng12/dbsrv12 server option" on page 240.
IsEccAvailable	Returns Yes if the ECC DLL is installed, and No otherwise.
IsFipsAvailable	Returns Yes if the FIPS DLL is installed, and No otherwise.
IsIQ	This property is reserved for system use. Do not change the setting of this property.
IsNetworkServer	Returns Yes if connected to a network database server, and No if connected to a personal database server.

Property name	Description
IsPortableDevice	Returns Yes if the database server is running on a laptop, notebook, or other portable device, and No otherwise. Yes is always returned on Windows Mobile. VMWare is not taken into account, so a database server running in a VM that is running on a laptop returns No. On Windows if it is not possible to determine whether the device is portable, this property returns NULL. This property always returns NULL on Unix.
IsRsaAvailable	Returns Yes if the RSA DLL is installed, and No otherwise.
IsRuntimeServer	Returns No for all versions of the database server.
IsService	Returns Yes if the database server is running as a service, and No otherwise.
Language	Returns the locale language for the server.
LastConnectionProperty	Returns the number that represents the last connection property.
LastDatabaseProperty	Returns the number that represents the last database property.
LastOption	Returns the number that represents the last connection property that corresponds to a database option.
LastServerProperty	Returns the number that represents the last server property.
LegalCopyright	Returns the copyright string for the software.
LegalTrademarks	Returns trademark information for the software.
LicenseCount	Returns the number of licensed seats or processors.
LicensedCompany	Returns the name of the licensed company.
LicensedUser	Returns the name of the licensed user.
LicenseType	Returns the license type. Can be networked seat (per-seat) or CPU-based.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
LivenessTimeout	Returns the client liveness timeout default. See "-tl dbeng12/dbsrv12 server option" on page 226.
LockedCursorPages	Returns the number of pages used to keep cursor heaps pinned in memory.
LockedHeapPages	Returns the number of heap pages locked in the cache.
MachineName	Returns the name of the computer running a database server. Typically, this is the computer's host name.
MainHeapBytes	Returns the number of bytes used for global server data structures.
MainHeapPages	Returns the number of pages used for global server data structures.
MapPhysicalMemoryEng	Returns the number of database page address space windows mapped to physical memory in the cache using Address Windowing Extensions.
MaxCacheSize	Returns the maximum cache size allowed, in kilobytes.
MaxConnections	Returns the maximum number of concurrent connections the server allows. For the personal server, this value defaults to 10. For the network server, this value defaults to about 32000. This value can be lowered using the -gm server option. See "-gm dbeng12/dbsrv12 server option" on page 188. Computer resources typically limit the number of connections to a network server to a lower value than the default.
MaxEventType	Returns the maximum valid event type ID.
MaxMessage	This property is deprecated. Returns the current maximum line number that can be retrieved from the database server messages window. This represents the most recent message displayed in the database server messages window.
MaxMultiProgrammingLevel	Returns the maximum number of tasks that the database server can process concurrently. See "Configuring the database server's multiprogramming level" on page 52.
MaxRemoteCapability	Returns the maximum valid capability ID.

Property name	Description
Message	This property is deprecated. Returns a line from the database server messages window, prefixed by the date and time the message appeared. The second parameter specifies the line number. The value returned by `PROPERTY( "message" )` is the first line of output that was written to the database server messages window. Calling `PROPERTY( "message", n )` returns the n-th line of server output (with zero being the first line). The buffer is finite, so as messages are generated, the first lines are dropped and may no longer be available in memory. In this case, NULL is returned.
MessageCategoryLimit	Returns the minimum number of messages of each severity and category that can be retrieved using the sa_server_messages system procedure. The default value is 400. See "sa_server_messages system procedure" [*SQL Anywhere Server - SQL Reference*].
MessageText	This property is deprecated. Returns the text associated with the specified line number in the database server messages window, without a date and time prefix. The second parameter specifies the line number.
MessageTime	This property is deprecated. Returns the date and time associated with the specified line number in the database server messages window. The second parameter specifies the line number.
MessageWindowSize	This property is deprecated. Returns the maximum number of lines that can be retrieved from the database server messages window.
MinCacheSize	Returns the minimum cache size allowed, in kilobytes.
MinMultiProgrammingLevel	Returns the minimum number of tasks that the server can process concurrently. See "Configuring the database server's multiprogramming level" on page 52.
MultiPacketsReceived	Returns the number of multi-packet requests received during client/server communications.
MultiPacketsSent	Returns the number of multi-packet responses sent during client/server communications.
MultiPageAllocs	Returns the number of multi-page cache allocations.
MultiProgrammingLevel	Returns the maximum number of concurrent tasks the server can process. Requests are queued if there are more concurrent tasks than this value. This can be changed with the -gn server option. See "-gn dbsrv12 server option" on page 189.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
Name	Returns the alternate name of the server used to connect to the database if one was specified, otherwise, returns the real server name. See "-sn dbsrv12 database option" on page 260. If the client is connected to a copy node and specified NodeType=COPY in the connection string, then the value of this property may be different than the database server name specified in the client connection string by the ServerName connection parameter. See "NodeType (NODE) connection parameter" on page 300.
NativeProcessorArchitecture	Returns a string that identifies the native processor type on which the software is running for platforms where a processor can be emulated (such as X86 on Win64). In all other cases, it returns the same value as the ProcessorArchitecture property. See "ProcessorArchitecture server property" on page 655. The value X86 can be returned for X86, IA86 or X86_64. This property does not return a value that indicates whether the operating system is 32-bit or 64-bit. Values can include: • 32-bit Windows, except Windows Mobile, - X86 • Windows Mobile - ARM • 64-bit Windows - X86_64 • Solaris - SPARC or X86_64 • AIX - PPC • MAC OS - X86 or X86_64 • HP - IA64 • Linux - X86 or X86_64 For more information on supported platforms, see http://www.sybase.com/detail?id=1002288.
NumLogicalProcessors	Returns the number of logical processors (including cores and hyperthreads) enabled on the server computer.
NumLogicalProcessorsUsed	Returns the number of logical processors the database server will use. On Windows, use the -gtc option to change the number of logical processors used. See "-gtc dbeng12/dbsrv12 server option" on page 196.
NumPhysicalProcessors	Returns the number of physical processors enabled on the server computer. This value is NumLogicalProcessors divided by the number of cores or hyperthreads per physical processor. On some non-Windows platforms, cores or hyperthreads may be counted as physical processors.

Property name	Description
NumPhysical-Processor-sUsed	Returns the number of physical processors the database server will use. The personal server is limited to one processor on some platforms. On Windows, you can use the -gt option to change the number of physical processors used by the network database server. See "-gt dbeng12/dbsrv12 server option" on page 195.
ObjectType	Returns the type of database object. This value is used by the SYSOBJECT system view. See "SYSOBJECT system view" [*SQL Anywhere Server - SQL Reference*].
OmniIdentifier	This property is reserved for system use. Do not change the setting of this property.
PacketsReceived	Returns the number of client/server communication packets received. This value is not updated for HTTP or HTTPS connections.
PacketsReceivedUncomp	Returns the number of packets that would have been received during client/server communications if compression was disabled. (This value is the same as the value for PacketsReceived if compression is disabled.)
PacketsSent	Returns the number of client/server communication packets sent. This value is not updated for HTTP or HTTPS connections.
PacketsSentUncomp	Returns the number of packets that would have been sent during client/server communications if compression was disabled. (This value is the same as the value for PacketsSent if compression is disabled.)
PageSize	Returns the size of the database server cache pages. This can be set using the -gp option, otherwise, it is the maximum database page size of the databases specified on the command line.
PeakCacheSize	Returns the largest value the cache has reached in the current session, in kilobytes.
Platform	Returns the operating system on which the software is running. For example, if you are running on Windows XP, this property returns WindowsXP.
PlatformVer	Returns the operating system on which the software is running, including build numbers, service packs, and so on.
ProcessCPU	Returns CPU usage for the database server process. Values are in seconds. This property is supported on Windows and Unix. This property is not supported on Windows Mobile. The value returned for this property is cumulative since the database server was started. The value will not match the instantaneous value returned by applications such as the Windows Task Manager or the Windows Performance Monitor.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
ProcessCPU-System	Returns system CPU usage for the database server process CPU. This is the amount of CPU time that the database server spent inside the operating system kernel. Values are in seconds. This property is supported on Windows and Unix. This property is not supported on Windows Mobile. The value returned for this property is cumulative since the database server was started. The value will not match the instantaneous value returned by applications such as the Windows Task Manager or the Performance Monitor.
ProcessCPUUser	Returns user CPU usage for the database server process. Values are in seconds. This excludes the amount of CPU time that the database server spent inside the operating system kernel. This property is supported on Windows and Unix. This property is not supported on Windows Mobile. The value returned for this property is cumulative since the database server was started. The value will not match the instantaneous value returned by applications such as the Windows Task Manager or the Performance Monitor.
ProcessorArchitecture	Returns a string that identifies the processor type that the current software was built for. Values include: • 32-bit Windows (except Windows Mobile) - X86 • 64-bit Windows - X86_64 • Windows Mobile - ARM • Solaris - SPARC or X86_64 • AIX - PPC • MAC OS - X86 • HP - IA64 • Linux - X86 or X86_64 For x86 and x64 platforms, this property returns X86 for 32-bit database servers and X86_64 for 64-bit database servers. For more information, see "NativeProcessorArchitecture server property" on page 653.
ProductName	Returns the name of the software.
ProductVersion	Returns the version of the software being run.
ProfileFilterConn	Returns the ID of the connection being monitored if procedure profiling for a specific connection is turned on. Otherwise, returns an empty string. You control procedure profiling by user with the sa_server_option procedure. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].

Property name	Description
ProfileFilterUser	Returns the name of the user being monitored if procedure profiling for a specific user is turned on. Otherwise, returns an empty string. You control procedure profiling by user with the sa_server_option procedure. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].
QueryHeapPages	Returns the number of cache pages used for query processing (hash and sort operations).
QueryMemActiveCurr	Returns the number of requests actively using query memory.
QueryMemActiveEst	Returns the database server's estimate of the steady state average of the number of requests actively using query memory.
QueryMemActiveMax	Returns the maximum number of requests that are actively allowed to use query memory.
QueryMemExtraAvail	Returns the amount of memory available to grant beyond the base memory-intensive grant.
QueryMemGrantBase	Returns the minimum amount of memory granted to all requests.
QueryMemGrantBaseMI	Returns the minimum amount of memory granted to memory-intensive requests.
QueryMemGrantExtra	Returns the number of query memory pages that can be distributed among active memory intensive beyond QueryMemGrantBaseMI.
QueryMemGrantFailed	Returns the total number of times a request waited for query memory, but failed to get it.
QueryMemGrantGranted	Returns the number of pages currently granted to requests.
QueryMemGrantRequested	Returns the total number of times any request attempted to acquire query memory.
QueryMemGrantWaited	Returns the total number of times any request waited for query memory.
QueryMemGrantWaiting	Returns the current number of requests waiting for query memory.
QueryMemPages	Returns the amount of memory that is available for query execution algorithms, expressed as a number of pages.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
QueryMemPercentOfCache	Returns the amount of memory that is available for query execution algorithms, expressed as a percent of maximum cache size.
QuittingTime	Returns the shutdown time for the server. If none is specified, the value is none.
RememberLastPlan	Returns Yes if the server is recording the last query optimization plan returned by the optimizer. See "-zp dbeng12/dbsrv12 server option" on page 247.
RememberLastStatement	Returns Yes if the server is recording the last statement prepared by each connection, and No otherwise. See "-zl dbeng12/dbsrv12 server option" on page 244.
RemoteCapability	Returns the remote capability name associated with a given capability ID.
RemoteputWait	Returns the number of remote put waits.
Req	Returns the number of times the server has been entered to allow it to handle a new request or continue processing an existing request.
RequestFilterConn	Returns the ID of the connection that logging information is being filtered for, otherwise, returns -1.
RequestFilterDB	Returns the ID of the database that logging information is being filtered for, otherwise, returns -1.
RequestLogFile	Returns the name of the request logging file. An empty string is returned if there is no request logging. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].
RequestLogging	Returns one of SQL, PLAN, HOSTVARS, PROCEDURES, TRIGGERS, OTHER, BLOCKS, REPLACE, ALL, or NONE, indicating the current setting for request logging. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].
RequestLogMaxSize	Returns the maximum size of the request log file. See "-zs dbeng12/dbsrv12 server option" on page 249.
RequestLogNumFiles	Returns the number of request log files being kept. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].
RequestsReceived	Returns the number of client/server communication requests or round trips. It is different from PacketsReceived in that multi-packet requests count as one request, and liveness packets are not included.

Property name	Description
RequestTiming	Returns Yes if request timing is turned on, and No otherwise. Request timing is turned on using the -zt database server option. See "-zt dbeng12/dbsrv12 server option" on page 250.
SendFail	Returns the number of times that the underlying communications protocols have failed to send a packet.
ServerEdition	Returns a space separated list of words describing the database server type. Values include: • Evaluation • Developer • Web • Evaluation • Standard • Advanced • Workgroup • OEM • Authenticated If you have a separate licence for any of the following features, then the appropriate string(s) are added to the license string that is returned: • **HighAvailability** See "SQL Anywhere high availability option" [*SQL Anywhere 12 - Introduction*]. • **InMemory** See "SQL Anywhere in-memory mode option" [*SQL Anywhere 12 - Introduction*]. • **ECC** See "SQL Anywhere security option" [*SQL Anywhere 12 - Introduction*]. • **FIPS** See "SQL Anywhere security option" [*SQL Anywhere 12 - Introduction*].
ServerName	Returns the name of the server for the current connection. You can use this value to determine which of the operational servers is currently acting as primary in a database mirroring configuration. See "Introduction to database mirroring" on page 945.
StartDBPermission	Returns the setting of the -gd server option, which can be one of DBA, all, or none. See "-gd dbeng12/dbsrv12 server option" on page 185.
StartTime	Returns the date/time that the server started.
StreamsUsed	Returns the number of database server streams in use.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
TcpIpAddresses	Returns a semicolon delimited list of the TCP/IP addresses the server is listening to for Command Sequence and TDS connections. For example: `(::1):2638;127.0.0.1:2638`
TempDir	Returns the directory in which temporary files are stored by the server.
ThreadDeadlocksAvoided	Returns the number of times a thread deadlock error was detected but not reported to client applications. When the database server starts, the value of this property is 0. To avoid returning thread deadlock errors, the database server dynamically increases the multiprogramming level. If the multiprogramming level cannot be increased, a thread deadlock error is returned to the client application and the ThreadDeadlocksReported property is incremented. The ThreadDeadlocksAvoided property is always 0 on the personal server because dynamic tuning of the multiprogramming level is disabled by default. For more information, see “Configuring the database server's multiprogramming level” on page 52.
ThreadDeadlocksReported	Returns the number of times a thread deadlock error was reported to client applications. When the database server starts, the value of this property is 0.
TimeZoneAdjustment	Returns the number of minutes that must be added to the Coordinated Universal Time (UTC) to display time local to the server.
TotalBuffers	Returns the total number of network buffers.
UniqueClientAddresses	Returns the number of unique client network addresses connected to a network server.
UnschReq	Returns the number of requests that are currently queued up waiting for an available server worker.
WebClientLogFile	Returns the name of the web service client log file. See “-zoc dbeng12/dbsrv12 server option” on page 246.
WebClientLogging	Returns a value that indicates whether web service client information is being logged to a file. See “-zoc dbeng12/dbsrv12 server option” on page 246.

Database properties

The following table lists properties available for each database on the database server.

You can use the DB_PROPERTY system function to retrieve the value for an individual property, or you can use the sa_db_properties system procedure to retrieve the values of all database properties. Property names are case insensitive.

Examples

To retrieve the value of a database property

- Use the DB_PROPERTY system function. For example, the following statement returns the page size of the current database:

```
SELECT DB_PROPERTY ( 'PageSize' );
```

To retrieve the values of all database properties

- Use the sa_db_properties system procedure:

```
CALL sa_db_properties;
```

See also

- "DB_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]
- "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*]
- "Database server properties" on page 644
- "Connection properties" on page 619

Descriptions

Property name	Description
AccentSensitive	Returns the status of the accent sensitivity feature. Returns Yes if the database is accent sensitive, No if it is not, or FRENCH if it is using French sensitivity rules.
Alias	Returns the database name.
AlternateMirrorServerName	Returns the alternate mirror server name associated with the database if one was specified. See "-sm dbsrv12 database option (deprecated)" on page 259.
AlternateServerName	Returns the alternate server name associated with the database if one was specified. See "-sn dbsrv12 database option" on page 260.
ArbiterState	Returns one of the following values: • **NULL** You are connected to a database that is not mirrored. • **connected** The arbiter server is connected to the primary server. • **disconnected** The arbiter server is not connected to the primary server. See "Introduction to database mirroring" on page 945.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
AuditingTypes	Returns the types of auditing currently enabled. See "auditing option" on page 514
Authenticated	Returns Yes if the database has been authenticated, or No if the database has not been authenticated.
BlankPadding	Returns On if the database has blank padding enabled. Otherwise, it returns Off.
CacheHits	Returns the number of database page lookups satisfied by finding the page in the cache.
CacheRead	The number of database pages that have been looked up in the cache.
CacheReadIndInt	Returns the number of index internal-node pages that have been read from the cache.
CacheReadIndLeaf	Returns the number of index leaf pages that have been read from the cache.
CacheReadTable	Returns the number of table pages that have been read from the cache.
CacheReadWorkTable	Returns the number of cache work table reads.
Capabilities	Returns the capability bits enabled for the database. This property is primarily for use by technical support.
CaseSensitive	Returns the status of the case sensitivity feature. Returns On if the database is case sensitive. Otherwise, it returns Off. In case sensitive databases, data comparisons are case sensitive. This setting does not affect the case sensitivity of identifiers. Passwords are always case sensitive. See "Case sensitivity" [*SQL Anywhere Server - SQL Usage*].
CatalogCollation	Returns the identifier for the collation used for the catalog. This property has extensions that you can specify when querying the property value. See "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].
CharSet	Returns the CHAR character set of the database. This property has extensions that you can specify when querying the property value. See "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].
CheckpointLogCommitToDisk	Returns the number of checkpoint log commits to disk.

Property name	Description
CheckpointLogPagesInUse	Returns the number of checkpoint log pages in use.
CheckpointLogPagesRelocated	Returns the number of relocated checkpoint log pages.
CheckpointLogPagesWritten	Returns the number of checkpoint log pages that have been written.
CheckpointLogSavePreimage	Returns the number of preimages of database pages that are being added to the checkpoint log.
CheckpointLogSize	Returns the size of the checkpoint log, in pages.
CheckpointLogWrites	Returns the number of writes to the checkpoint log.
CheckpointUrgency	Returns the time that has elapsed since the last checkpoint, as a percentage of the checkpoint time setting of the database.
Checksum	Returns On if global checksums are enabled for the database (a checksum is created for each database page). Otherwise, returns Off. Checksums are always present for critical pages.
Chkpt	Returns the number of checkpoints that have been performed.
ChkptFlush	Returns the number of ranges of adjacent pages written out during a checkpoint.
ChkptPage	Returns the number of transaction log checkpoints.
CleanablePagesAdded	Returns the number of pages marked to be cleaned since database server startup.
CleanablePagesCleaned	Returns the number of database pages cleaned since database server startup.
CleanableRowsAdded	Returns the number of rows marked to be deleted since database server startup.
CleanableRowsCleaned	Returns the number of shadow table rows deleted since database server startup.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
Collation	Returns the collation used by the database. For a list of available collations, see "Supported and alternate collations" on page 423. This property has extensions that you can specify when querying the property value. See "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].
CommitFile	Returns the number of times the server has forced a flush of the disk cache. On Windows, the disk cache doesn't need to be flushed if unbuffered (direct) I/O is used.
ConnCount	Returns the number of connections to the database. The property value does not include connections used for firing events or other internal operations, but it does include connections used for firing events and external environment support. If you want to obtain an accurate count of the number of licensed connections in use, you can execute the following statement: `SELECT COUNT( * ) FROM sa_conn_info( )`
ConnPoolCachedCount	Returns the number of connections disconnected by the application but cached for connection pooling.
ConnPoolHits	Returns the number of reused pooled connections.
ConnPoolMisses	Returns the number of pooled connections which were cached but could not be reused.
ConnsDisabled	Returns On if connections to the current database are disabled, otherwise, returns Off.
CurrentRedoPos	Returns the current offset in the transaction log file where the next database operation is to be logged.
CurrIO	Returns the current number of file I/Os that were issued by the server but haven't yet completed.
CurrRead	Returns the current number of file reads that were issued by the server, but haven't yet completed.
CurrWrite	Returns the current number of file writes that were issued by the server, but haven't yet completed.
DatabaseCleaner	Returns On or Off to indicate whether the database cleaner is enabled.
DBFileFragments	Returns the number of database file fragments. This property is supported on Windows.

Property name	Description
DiskRead	Returns the number of pages that have been read from disk.
DiskReadHint	Returns the number of disk read hints.
DiskReadHint-Pages	Returns the number of disk read hint pages.
DiskReadIn-dInt	Returns the number of index internal-node pages that have been read from disk.
DiskReadIn-dLeaf	Returns the number of index leaf pages that have been read from disk.
DiskReadTable	Returns the number of table pages that have been read from disk.
DiskRead-WorkTable	Returns the number of disk work table reads.
DiskRetryR-eadScatter	Returns the number of disk read retries for scattered reads.
DiskSyncRead	Returns the number of disk reads issued synchronously.
DiskSyncWrite	Returns the number of writes issued synchronously.
DiskWaitRead	Returns the number of times the database server waited for an asynchronous read.
DiskWaitWrite	Returns the number of times the database server waited for an asynchronous write.
DiskWrite	Returns the number of modified pages that have been written to disk.
DiskWriteHint	Returns the number of disk write hints.
DiskWriteHint-Pages	Returns the number of disk gather write hints.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
DriveBus	Returns the type of bus to which the database file is connected. The value is The value is one of the following: • Unknown • SCSI • ATAPI • ATA • 1394 • SSA • Fibre • USB • RAID • iSCSI • SAS • SATA • SD • MMC • Virtual • FileBackedVirtual If the bus type cannot be determined, this property returns NULL. The value of this property for the primary dbspace is recorded in the SYSHISTORY view. See "SYSHISTORY system view" [*SQL Anywhere Server - SQL Reference*].
DriveModel	Returns the model number of the drive on which the database is located. The value of this property for the primary dbspace is recorded in the SYSHISTORY view. See "SYSHISTORY system view" [*SQL Anywhere Server - SQL Reference*]. If the drive model cannot be determined, this property returns NULL.

Property name	Description
DriveType	Returns the type of drive on which the database file is located. The value is one of the following: • CD • FIXED • RAMDISK • REMOTE • REMOVABLE • UNKNOWN On Unix, depending on the version of Unix and the type of drive, it may not be possible to determine the drive type. In these cases UNKNOWN is returned. This property has extensions that you can specify when querying the property value. See "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].
Encryption	Returns the type of encryption used for database or table encryption, one of None, Simple, AES, AES256, AES_FIPS, or AES256_FIPS.
EncryptionScope	Returns the part of the database, if any, that can be encrypted. The value is one of the following: TABLE, DATABASE, or NONE. TABLE indicates that table encryption is enabled. DATABASE indicates that the whole database is encrypted. NONE indicates that table encryption is not enabled, and the database is not encrypted.
ExprCacheAbandons	Returns the number of time that the expression cache was completely abandoned because the hit rate was too low.
ExprCacheDropsToReadOnly	Returns the number of times that the expression cache dropped to read-only status because the hit rate was low.
ExprCacheEvicts	Returns the number of evictions from the expression cache.
ExprCacheHits	Returns the number of hits in the expression cache.
ExprCacheInserts	Returns the number of values inserted into the expression cache.
ExprCacheLookups	Returns the number of lookups performed in the expression cache.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
ExprCacheResumesOfReadWrite	Returns the number of times that the expression cache resumed read-write status because the hit rate increased.
ExprCacheStarts	Returns the number of times the expression cache was started.
ExtendDB	Returns the number of pages by which the database file has been extended.
ExtendTempWrite	Returns the number of pages by which temporary files have been extended.
File	Returns the file name of the database root file, including path. This property has extensions that you can specify when querying for property value. See "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].
FileSize	Returns the file size of the system dbspace, in pages. This property has extensions that you can specify when querying for property value. See "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].
FreePages	Returns the number of free pages in the system dbspace. The FreePages property is only supported on databases created with version 8.0.0 or later. This property has extensions that you can specify when querying for property value. See "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].
FullCompare	Returns the number of comparisons that have been performed beyond the hash value in an index.
GetData	Returns the number of GETDATA requests.
GlobalDBID	Returns the value of the global_database_id option used to generate unique primary key values in a replication environment.
HasCollationTailoring	Returns a value indicating whether collation tailoring was specified when the database was created. Possible values are On or Off.
HasEndianSwapFix	Returns a value indicating whether the database supports both big-endian and little endian UTF-16 encoding on all platforms, regardless of the endianness of the platform. Possible values are On or Off.
HashForcedPartitions	Returns the number of times that a hash operator was forced to partition because of competition for memory.
HashRowsFiltered	Returns the number of probe rows rejected by bit-vector filters.

Property name	Description
HashRowsPartitioned	Returns the number of rows written to hash work tables.
HashWorkTables	Returns the number of work tables created for hash-based operations.
HasNCHARLegacyCollationFix	Returns one of the following values: • **ON** For all databases created using version 11 or later, and databases created by a version 10 database server with the legacy collation fix and that use a legacy NCHAR collation. • **OFF** For databases created using a version 10 database server without the legacy collation fix, or databases created using a version 10 database server that do not use a legacy NCHAR collation.
HasTornWriteFix	Returns a value that indicates whether the database has a fixed file format to allow recovery from partial writes.
HttpConnPoolCachedCount	Returns the absolute count of cached database connections within all pools.
HttpConnPoolHits	Returns the rate of connections reused by the same service.
HttpConnPoolMisses	Returns the rate of new connections that were allocated when a connection could not be accessed from a pool. This property only counts HTTP requests to services that use connection pooling. At the time of access, a connection may not have been available due to a small pool size whose oldest connection did not fit the steal criteria.
HttpConnPoolSteals	Returns the rate of connections taken by services where the connection belonged to another service. The service steals a connection from another service if the criteria is met for an HTTP request for a service not having any direct reusable connections and the pool size and age of the least used connection. A connection is allocated for the service and Http_Conn_Pool_Misses is incremented instead if the pool criteria is not met.
IdentitySignature	This property is reserved for system use. Do not change the setting of this property.
IdleCheck	Returns the number of times that the server's idle thread has become active to do idle writes, idle checkpoints, and so on.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
IdleChkpt	Returns the number of checkpoints completed by the server's idle thread. An idle checkpoint occurs whenever the idle thread writes out the last dirty page in the cache.
IdleChkTime	Returns the number of hundredths of a second spent checkpointing during idle I/O.
IdleWrite	Returns the number of disk writes that have been issued by the server's idle thread.
IndAdd	Returns the number of entries that have been added to indexes.
IndLookup	Returns the number of entries that have been looked up in indexes.
IOParallelism	Returns the estimated number of simultaneous I/O operations supported by the dbspace. This property has extensions that you can specify when querying the property value. See "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].
IOToRecover	Returns the estimated number of I/O operations required to recover the database.
IQStore	This property is reserved for system use. Do not change the setting of this property.
JavaVM	Returns the Java VM the database server uses to execute Java in the database.
Language	Returns a comma-separated list of languages known to be supported by the database collation. The languages are in two-letter ISO format. If the language isn't known, the return value is NULL. For a list of the two-letter ISO format language names and the language they correspond to, see "Understanding the locale language" on page 417.
LastCheckpointTime	Returns the date and time, in millisecond, of the most recent checkpoint.
LockCount	Returns the number of locks held by the database.
LockTablePages	Returns the number of pages used to store lock information.
LogFileFragments	Returns the number of log file fragments. This property is supported on Windows.
LogFreeCommit	Returns the number of redo free commits. A redo free commit occurs when a commit of the transaction log is requested but the log has already been written (so the commit was done for free).
LogMirrorName	Returns the file name of the transaction log mirror, including path.
LogName	Returns the file name of the transaction log, including path.

Property name	Description
LogWrite	Returns the number of pages that have been written to the transaction log.
LTMGeneration	This property is for internal use only.
LTMTrunc	This property is for internal use only.
MaxIO	Returns the maximum value that CurrIO has reached.
MaxRead	Returns the maximum value that CurrRead has reached.
MaxWrite	Returns the maximum value that CurrWrite has reached.
MirrorMode	Returns NULL if database mirroring is not in use, synchronous if the mirroring mode specified with the -xp option is synchronous, and asynchronous otherwise.
MirrorRole	Returns one of the following values: • **primary** The primary database server or mirror server. • **mirror** The mirror server. • **copy** A read-only mirror server that obtains its log pages from the current primary server, the current mirror server, or another read-only node. A copy node cannot become the primary server. A copy node may also be referred to as a child node, since it has a parent from which it receives log pages. See "Introduction to database mirroring" on page 945.
MirrorServerState	Returns one of the following values: • **connected** The mirror server is connected. • **disconnected** The mirror server is not connected or has not yet read all the primary server's log pages. • **null** The database is not mirrored. See "Introduction to database mirroring" on page 945.
MirrorServerWaits	Returns the number of times the database server waited more than 500 milliseconds when sending log pages to copy servers.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
MirrorState	Returns one of the following values: • **null** You are connected to a database that is not mirrored. • **synchronizing** The mirror server is not connected or has not yet read all the primary server's log pages. This value is also returned if the synchronization mode is asynchronous. • **synchronized** The mirror server is connected and has all changes that have been committed on the primary server. See "Introduction to database mirroring" on page 945.
MultiByteCharSet	Returns On if the database uses a multibyte character set, such as UTF-8. Otherwise, returns Off. For more information, see "Multibyte character sets" on page 407.
Name	Returns the database name (identical to "Alias database property" on page 660).
NcharCharSet	Returns the NCHAR character set of the database.
NcharCollation	Returns the name of the collation used for NCHAR data. This property has extensions that you can specify when querying the property value. See "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].
NextScheduleTime	Returns the next scheduled execution time for a specified event; query this property using the DB_EXTENDED_PROPERTY function. See "DB_EXTENDED_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].
OptionWatchAction	Returns the action that is taken when an attempt is made to set a database option that is included in the OptionWatchList property. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].
OptionWatchList	Returns the list of database options being monitored by the database server. See "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].
PageRelocations	Returns the number of relocatable heap pages that have been read from the temporary file.
PageSize	Returns the page size of the database, in bytes.

Property name	Description
PartnerState	Returns one of the following values: • **NULL** You are connected to a database that is not mirrored. • **connected** The mirror server is connected to the primary server. • **disconnected** The mirror server is not connected to the primary server. See "Introduction to database mirroring" on page 945.
Prepares	Returns the number of statement preparations performed for the database.
ProcedurePages	Returns the number of relocatable heap pages that have been used for procedures.
ProcedureProfiling	Returns On if procedure profiling is turned on for the database. Otherwise, returns Off.
QueryBypassed	Returns the number of requests reused from the plan cache.
QueryBypassedCosted	Returns the number of requests processed by the optimizer bypass using costing.
QueryBypassedHeuristic	Returns the number of requests processed by the optimizer bypass using heuristics.
QueryBypassedOptimized	Returns the number of requests initially processed by the optimizer bypass and subsequently fully optimized by the SQL Anywhere optimizer.
QueryCachedPlans	Returns the number of cached execution plans across all connections.
QueryCachePages	Returns the number of pages used to cache execution plans.
QueryDescribedBypass	Returns the number of describe requests processed by the optimizer bypass.
QueryDescribedOptimizer	Returns the number of describe requests processed by the optimizer.
QueryJHToJNLOptUsed	Returns the number of times a hash join was converted to a nested loops join.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
QueryLowMemoryStrategy	Returns the number of times the server changed its execution plan during execution as a result of low memory conditions. The strategy can change because less memory is available than the optimizer estimated, or because the execution plan required more memory than the optimizer estimated.
QueryOpened	Returns the number of OPEN requests for execution.
QueryOptimized	Returns the number of requests fully optimized.
QueryReused	Returns the number of reused query plans.
QueryRowsFetched	Returns the number of rows that have been read from base tables, either by a sequential scan or an index scan, for this database.
QueryRowsMaterialized	Returns the number of rows written to work tables during query processing.
ReadOnly	Returns On if the database is being run in read-only mode. Otherwise, returns Off.
ReceivingTracingFrom	Returns the name of the database from which the tracing data is coming. Returns a blank string if tracing is not attached.
RecoveryUrgency	Returns an estimate of the amount of time required to recover the database as a percentage of the recovery time setting of the database. See "-gr dbeng12/dbsrv12 server option" on page 194 and "How the database server decides when to checkpoint" on page 924.
RecursiveIterations	Returns the number of iterations for recursive unions.
RecursiveIterationsHash	Returns the number of times recursive hash join used a hash strategy.
RecursiveIterationsNested	Returns the number of times recursive hash join used a nested loops strategy.
RecursiveJNLMisses	Returns the number of index probe cache misses for recursive hash join.
RecursiveJNLProbes	Returns the number of times recursive hash join attempted an index probe.
RelocatableHeapPages	Returns the number of pages used for relocatable heaps (cursors, statements, procedures, triggers, views, and so on.).
RemoteTrunc	Returns the minimal confirmed log offset for the SQL Remote Message Agent.

Property name	Description
RollbackLog-Pages	Returns the number of pages in the rollback log.
SendingTra-cingTo	Returns the connection string where the tracing data is being sent. Returns a blank string if tracing is not attached.
SnapshotCount	Returns the number of snapshots associated with the database.
SnapshotIsola-tionState	Returns one of the following values: • **On** snapshot isolation is enabled for the database. • **Off** snapshot isolation is disabled for the database. • **in_transition_to_on** snapshot isolation will be enabled once the current transactions complete. • **in_transition_to_off** snapshot isolation will be disabled once the current transactions complete. See "allow_snapshot_isolation option" on page 506.
SortMerge-Passes	Returns the number of merge passes used during sorting.
SortRowsMate-rialized	Returns the number of rows written to sort work tables.
SortRunsWrit-ten	Returns the number of sorted runs written during sorting.
SortSorte-dRuns	Returns the number of sorted runs created during run formation.
SortWorkTa-bles	Returns the number of work tables created for sorting.
StatementDe-scribes	Returns the total number of statements processed by DESCRIBE requests.
StatementPos-tAnnotates	Returns the number of statements processed by the semantic query transformation phase.
StatementPos-tAnnotatesSim-ple	Returns the number of statements processed by the semantic query transformation phase, but which skipped some of the semantic transformations.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Property name	Description
StatementPostAnnotatesSkipped	Returns the number of statements that have completely skipped the semantic query transformation phase.
SynchronizationSchemaChangeActive	Returns On if an active connection issued a START SYNCHRONIZATION SCHEMA CHANGE but has not issued a STOP SYNCHRONIZATION SCHEMA CHANGE. See "START SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*] and "STOP SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*].
SyncTrunc	Returns the minimal confirmed log offset for the MobiLink client dbmlsync executable.
TempFileName	Returns the file name of the database temporary file, including path.
TempTablePages	Returns the number of pages in the temporary file used for temporary tables.
TriggerPages	Returns the number of relocatable heap pages used for triggers.
VersionStorePages	Returns the number of pages in the temporary file that are being used for the row version store when snapshot isolation is enabled.
ViewPages	Returns the number of relocatable heap pages used for views.
WriteChecksum	Returns On if the database server adds checksums to pages before they are written out; otherwise, returns Off. See "Using checksums to detect corruption" on page 928.
XPathCompiles	Returns the number of times any XPath query (using the openxml procedure) was compiled by the database server since database server startup. See "openxml system procedure" [*SQL Anywhere Server - SQL Reference*].

SQL Anywhere size and number limitations

The following table lists the physical limitations on size and number of objects in a SQL Anywhere database. Typically, the memory, CPU, and disk drive of the computer are the most limiting factors.

Item	Limitation
Database size	13 files per database. For each file, the largest file allowed by operating system and file system

Item	Limitation
Dbspace size	2^{28} x page size
Temporary file size	2^{28} x page size
Field size	2 GB
File size (FAT 12)	16 MB
File size (FAT 16)	2 GB
File size (FAT 32)	4 GB
File size for NTFS, HP-UX 11.0 and later, Solaris 2.6 and later, Linux 2.4 and later)	• 512 GB for 2 KB pages • 1 TB for 4 KB pages • 2 TB for 8 KB pages
File size (all other platforms and file systems)	2 GB
Maximum cache size (non-AWE cache) (Windows XP Home Edition, Windows XP Professional, Windows Server 2003 Web Edition, Windows Server 2003 Standard Edition, Windows 2008, Windows 7)	1.8 GB
Maximum cache size (non-AWE cache) (Windows Server 2003 Enterprise Edition, Windows Server 2003 Datacenter Edition, Windows Vista Ultimate, Windows Vista Enterprise, Windows Vista Business, Windows Vista Home Premium, Windows Vista Home Basic)	2.7 GB
Maximum cache size (AWE cache) (Windows XP Home Edition, Windows XP Professional, Windows Server 2003 Web Edition, Windows Server 2003 Standard Edition, Windows Server 2003 Enterprise Edition, Windows Server 2003 Datacenter Edition)	100% of all available memory - 128 MB
Maximum cache size (Windows Mobile)	Limited by available memory on the device
Maximum cache size (Unix—Solaris, x86 Linux, IBM AIX, HP)	2 GB for 32-bit servers
Maximum cache size (Win 64)	Limited by physical memory on 64-bit servers
Maximum cache size (Itanium HP-UX)	Limited by physical memory on 64-bit servers

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Item	Limitation
Maximum index entry size	No limit
Number of databases per server	255
Number of columns per table	45000 Note: An excessive number of columns, although allowed, can affect performance.
Number of nullable columns per table	min(45000, (page size - overhead) * 8)
Number of columns in a procedure result set	45000
Number of columns in a SELECT list	100000
Number of columns in a GROUP BY list	100000
Number of columns in a GROUP BY with grouping sets	64
Number of columns in a CUBE	15
Number of distinct grouping sets	32768
Length of DEFAULT for a column	32768
Length of COMPUTE for a column	32768
Length of DEFAULT for procedure parameters	32768
Length of DEFAULT for a user-defined domain	32768
Length of check constraints	2 GB
Number of indexes per table	2^{32}
Number of rows per database	4096 x 2^{28} x 13
Number of rows per table	4096 x 2^{28}
Number of tables per database	2^{32} - 2^{20} - 1 = 4293918719
Number of temporary tables per connection	2^{20} = 1048576
Number of tables referenced per transaction	No limit
Number of stored procedures per database	2^{32} - 1 = 4294967295
Number of concurrent statements per database server	20 x *number-of-database-connections* + 65534

Item	Limitation
Number of events per database	2^{31} - 1 = 2147483647
Number of triggers per database	2^{32} - 1 = 4294967295
Row size	Limited by file size
Table size	Maximum file size. User-created indexes for the table can be stored separately from the table
Strings	2 GB
Binary data types	2 GB
Identifiers (including user IDs, table names, and column names)	128 bytes
Passwords	255 bytes
Database server names	250 bytes (TCP/IP and shared memory See "-n dbeng12/dbsrv12 server option" on page 206 and "ServerName (Server) connection parameter" on page 306.
Database names	250 bytes See "-n dbeng12/dbsrv12 database option" on page 257.

SQL Anywhere hardware requirements

For information about hardware requirements, see www.sybase.com/detail?id=1069662.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Administering your database

This section describes how to use the tools included with SQL Anywhere to administer your database.

SQL Anywhere graphical administration tools

Using Sybase Central

Sybase Central is a graphical tool for managing your database servers, databases, and the objects they contain.

From within Sybase Central, you can get additional information about using and configuring Sybase Central by choosing **Help** » **Sybase Central**.

Sybase Central key features

- **Easy command access** The **File** menu in Sybase Central automatically updates when you select an object, providing commands related directly to that object. You can also right-click an object to access these commands.
- **Task wizards** If you want to add a new object, Sybase Central provides you with wizards that walk you through the task step by step.
- **Drag-and-drop functionality** Sybase Central supports drag-and-drop functionality for many operations. For example, if you want to copy tables to a different database, you can click and drag them to that location. See "Copying database objects in the SQL Anywhere 12 plug-in" on page 691.
- **Keyboard shortcuts** Many commonly-used commands have keyboard shortcuts; these shortcuts are listed beside the command names in the menus. See "Sybase Central keyboard shortcuts" on page 684.
- **Plug-in support** You can manage a variety of database products and tools by using plug-ins. From within Sybase Central, you can get additional information about using and configuring a plug-in by choosing the plug-in name from the **Help** menu.

Plug-ins

Each product is managed by a separate plug-in. The plug-ins for these products must be registered and loaded before you can use the products in Sybase Central. When you install a product, its plug-in is automatically registered and loaded.

SQL Anywhere includes Sybase Central plug-ins for the following products:

- SQL Anywhere databases
- UltraLite databases
- MobiLink synchronization
- QAnywhere messaging
- Relay Server

The plug-in files are found in the following location in your SQL Anywhere installation:

Plug-in	File name and location
SQL Anywhere 12	*install-dir\java\saplugin.jar*
MobiLink 12	*install-dir\java\mlplugin.jar*
UltraLite 12	*install-dir\java\ulplugin.jar*
QAnywhere 12	*install-dir\java\qaplugin.jar*
Relay Server 12	*install-dir\java\rsplugin.jar*

For more information about using the plug-ins included with SQL Anywhere, see:

- SQL Anywhere: "Using the SQL Anywhere 12 plug-in" on page 691
- MobiLink: "MobiLink Plug-in for Sybase Central" [*MobiLink - Getting Started*]
- UltraLite: "Create a database with the Create Database Wizard" [*UltraLite - Database Management and Reference*] and "Working with UltraLite databases" [*UltraLite - Database Management and Reference*]
- QAnywhere: "QAnywhere 12 plug-in" [*QAnywhere*]
- Relay Server: "Relay Server plug-in for Sybase Central" [*Relay Server*]

Deploying Sybase Central

Subject to your license agreement, you can deploy SQL Anywhere administration tools, including Sybase Central.

For information about deploying Sybase Central with your application, see "Deploying administration tools" [*SQL Anywhere Server - Programming*].

Starting Sybase Central

This section provides steps for starting Sybase Central and using the **SQL Anywhere 12** plug-in to connect to the SQL Anywhere sample database.

To start Sybase Central and connect to the sample database (Windows)

1. From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. Choose **Connections** » **Connect With SQL Anywhere 12**.
3. From the **Action** dropdown list, select **Connect With An ODBC Data Source**.
4. Select **ODBC Data Source Name** and then in the box below, type **SQL Anywhere 12 Demo**.
5. Click **Connect**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Mac OS X note
The administration tools only run on Intel-based Macintosh computers with 64-bit processors supported by the Apple JDK 1.6 (Mac OS X 10.5.2 or later). See http://www.sybase.com/detail?id=1061806.

To start Sybase Central and connect to the sample database (Mac OS X)

1. In the Finder, double-click **Sybase Central** in */Applications/SQLAnywhere12*.
2. Choose **Connections** » **Connect With SQL Anywhere 12**.
3. From the **Action** dropdown list, select **Connect With An ODBC Data Source**.
4. Select **ODBC Data Source Name**, and then in the box below, type **SQL Anywhere 12 Demo**.
5. Click **Connect**.

Note
The following steps assume that you have already sourced the SQL Anywhere utilities. See "Setting environment variables on Unix and Mac OS X" on page 377.

To start Sybase Central and connect to the sample database (Unix command line)

1. In a terminal session, enter the following command:

   ```
   scjview
   ```

   Sybase Central opens.
2. Choose **Connections** » **Connect With SQL Anywhere 12**.
3. From the **Action** dropdown list, select **Connect With An ODBC Data Source**.
4. Select **ODBC Data Source Name**, and then in the box below, type **SQL Anywhere 12 Demo**.
5. Click **Connect**.

The following steps can be used if you are using a version of Linux that supports the Linux Applications menu and if you chose to install the menu items when you installed SQL Anywhere.

To start Sybase Central and connect to the sample database (Linux Applications menu)

1. From the **Applications** menu, choose **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.

   Sybase Central opens.
2. Choose **Connections** » **Connect With SQL Anywhere 12**.
3. From the **Action** dropdown list, select **Connect With An ODBC Data Source**.
4. Select **ODBC Data Source Name**, and then in the box below, type **SQL Anywhere 12 Demo**.

5. Click **Connect**.

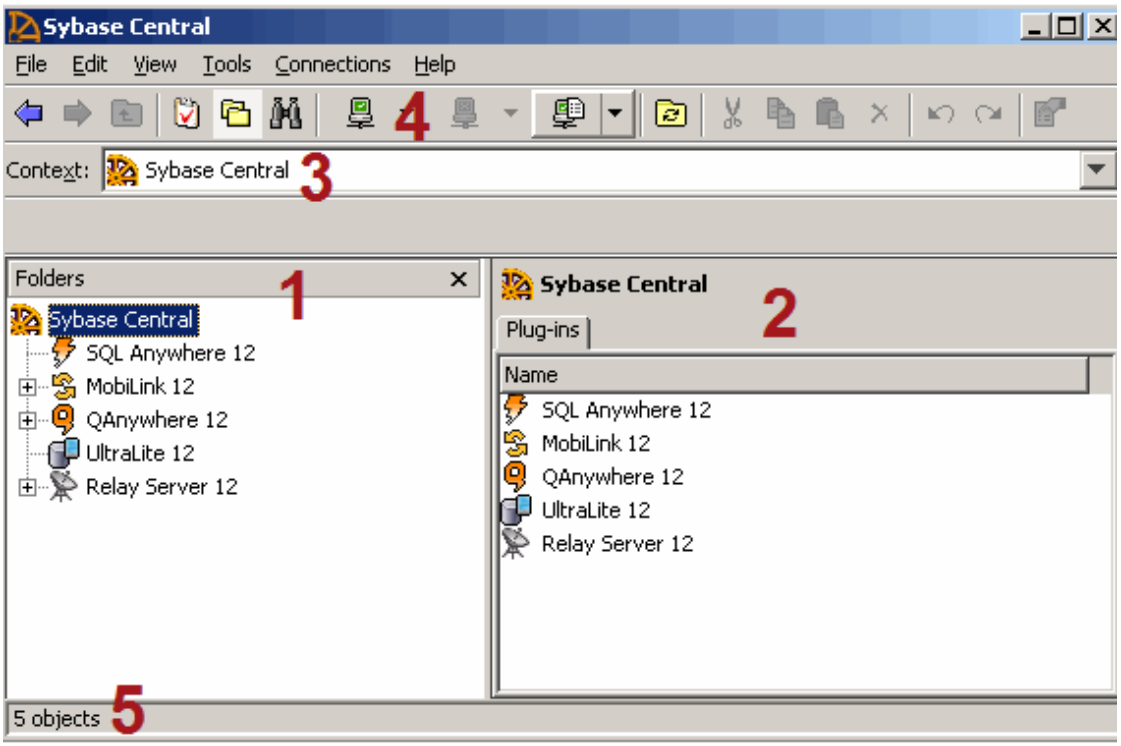
See also

- "Working with the Connect window" on page 92

Navigating Sybase Central

This section explains how to navigate the Sybase Central user interface.

The Sybase Central main window:



The Sybase Central window is split into two vertically-aligned panes.

1: Left pane

You can choose whether you want the left pane to display the:

- **Folders pane** This pane displays a hierarchical view of database objects.

  Folders shows only the containers in the object tree; it does not show objects that are not containers of other objects. For example, the left pane may show a Columns folder (a container), but not the columns themselves because they are items, and appear in the right pane instead.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Tasks pane** This pane displays a task list for the currently-selected database object.
- **Search pane** This pane allows you to search for objects in a plug-in. See "Searching databases in Sybase Central" on page 684.

To change the layout in the left pane

1. Start Sybase Central.
2. From the **View** menu, choose **Tasks**, **Folders**, or **Search** to view the task list, folders list, or search feature respectively.

2: Right pane

The right pane shows the contents of the currently selected container. The right pane has tabs that display the contents of the container that is selected in the left pane, and other information about the selected container.

You can configure the columns that appear on a tab in the right pane by choosing **View** » **Choose Columns**.

You can change the appearance of the right pane in the **Options** window (accessed through the **Tools** menu).

Once you connect to a database or database server, you can administer it by navigating and selecting its objects in the main window.

3: Toolbar

The main window toolbar provides you with buttons for common commands. To show or hide the toolbar, from the **View** menu, choose **Toolbars** » **Standard Toolbars**. With the main toolbar, you can:

- navigate through the object folders
- connect to or disconnect from a database, database server, or product plug-in
- show the **Tasks**, **Folders**, or **Search** pane
- access the **Connection Profiles** window (also accessible from the **Tools** menu)
- refresh the view of the current folder
- cut, copy, paste, and delete objects
- undo or redo actions
- view the properties window for a selected object

4: Context dropdown list

The **Context** dropdown list, which appears below the toolbar, lets you navigate the object folders for a plug-in.

5: Status bar

The status bar, which appears at the bottom of the main window, shows a brief summary of menu commands as you navigate through the menus. To show or hide the status bar, choose **View** » **Status Bar**.

Searching databases in Sybase Central

Sybase Central allows you to search a database for a specified database object, or to search for a string within the SQL of a database object.

To search for a specified object

1. In Sybase Central, choose **View** » **Search Pane**.

   The **Search** pane appears in the left pane.

2. Configure options for the search.

3. Click **Search**.

   The search results appear in **Results** in the left pane.

4. Double-click a result to open it in the right pane.

The **SQL Anywhere 12** plug-in allows you to search:

- **Search In SQL (Procedures, Events, Functions, and Triggers)** Select this option to include the SQL of procedures, events, functions and triggers in the search.
- **Search Dynamic Properties (Connections, Statistics, Locks)** Select this option to include dynamic properties, such as connected users, SQL Remote statistics, table locks, and table page usage information in the search.

The MobiLink 12 plug-in allows you to search:

- **Search In Scripts** Select this option to include synchronization scripts in the search.

Sybase Central keyboard shortcuts

Sybase Central provides the following keyboard shortcuts.

Function key	Description
Alt+Enter	Opens the properties window for the selected item.
Ctrl+C	Copies the selection to the clipboard.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Function key	Description
Ctrl+V	Inserts the clipboard contents.
Ctrl+X	Cuts the selection and moves it to the clipboard.
Delete	Deletes the selection.
F1	Opens the Sybase Central help.
F5	Refreshes the contents of the selected folder.
F9	Opens the **Connection Profiles** window.
F11	Opens the **Connection** menu if there are multiple plug-ins loaded. If only one plug-in is loaded, pressing F11 opens the **Connect** window for that plug-in.
F12	Disconnects when there is only one connection in Sybase Central. When there is more than one connection, pressing F12 opens the **Disconnect** window where you can select the connection you want to disconnect.
Shift+F10	Opens the popup menu for the selected object.

Using the Code Editor

The Code Editor appears as a **SQL** tab in the right pane of Sybase Central, as a separate window in Sybase Central, and as the **SQL Statements** pane in Interactive SQL.

Beyond the standard text editing functions, the Code Editor provides the following functionality:

- a toolbar and status bar
- automatic syntax highlighting
- language-sensitive indenting
- the ability to find and replace text
- the ability to open from and save to files (the availability of this functionality depends on the plug-in you are using)
- the ability to print the code
- text completion when typing code

To open the Code Editor in a separate window (Sybase Central)

1. Select a database object, such as a stored procedure, view, or trigger, in the left or right pane.
2. Choose **File** » **Edit In New Window**, or press Ctrl+E.

Customizing the Code Editor

You can customize the display characteristics of the Code Editor using the **Options** window. This window lets you change settings for the foreground and background colors, and the overall Code Editor appearance. All changes you make persist between sessions.

To set Code Editor settings when editing on the SQL tab

1. From the **File** menu, choose **Customize Editor**.
2. Configure the settings on the various tabs. Click **OK**.

To set Code Editor settings when editing in a separate window

1. In the Code Editor, choose **Tools** » **Options**.
2. Configure the settings on the various tabs. Click **OK**.

Code Editor keyboard shortcuts

Sybase Central provides the following keyboard shortcuts for the Code Editor.

Function key	Description
Alt+F4	Closes the Code Editor (if a separate window) or closes Sybase Central if you are editing text in the right pane of Sybase Central.
Backspace	Deletes the selection. If nothing is selected, pressing Backspace deletes the character to the left of the cursor.
Ctrl+]	Moves the cursor to the matching brace. Use this shortcut to match parentheses, braces, brackets, and angle brackets.
Ctrl+A	Selects the entire contents of the Code Editor window.
Ctrl+Backspace	Deletes the word to the left of the cursor.
Ctrl+C	Copies the selected text to the clipboard.
Ctrl+Delete	Deletes the word to the right of the cursor.
Ctrl+End	Moves the cursor to the bottom of the Code Editor window.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Function key	Description
Ctrl+F	Opens the **Find/Replace** window where you can search for and replace the specified text if you have not searched for text in the current window. Otherwise, this searches for the next occurrence of the specified text.
Ctrl+F3	Finds the next occurrence of the currently-selected text.
Ctrl+G	Opens the **Go To** window where you can specify the line location you want to go to within the Code Editor window.
Ctrl+Home	Moves the cursor to the top of the Code Editor window.
Ctrl+L	Deletes the current line.
Ctrl+Left Arrow	Moves the cursor back one word.
Ctrl+N	Clears the contents of the Code Editor window and closes the current file (if any). This shortcut cannot be used from the **SQL** tab in the right pane of Sybase Central.
Ctrl+O	Opens a file when the Code Editor is open as a separate window. This shortcut cannot be used from the **SQL** tab in the right pane of Sybase Central.
Ctrl+P	Prints the contents of the Code Editor window. You can configure the appearance of the printed text: from the **Tools** menu, choose **Options**, and then click the **Print** tab.
Ctrl+Right Arrow	Moves the cursor forward one word.
Ctrl+S	Saves the contents of the Code Editor window.
Ctrl+Shift+]	Extends the selection to the matching brace. Use this shortcut to match parentheses, braces, brackets, and angle brackets.
Ctrl+Shift+End	Extends the selection to the end of the code.
Ctrl+Shift+F3	Find the previous occurrence of the currently-selected text.
Ctrl+Shift+Home	Extends the selection to the beginning of the code.
Ctrl+Shift+L	Deletes the current line.
Ctrl+Shift+Left Arrow	Extends the selection back one word.
Ctrl+Shift+Right Arrow	Extends the selection forward one word.

Function key	Description
Ctrl+Shift+U	Changes the selection to uppercase characters.
Ctrl+Shift+Period (.)	Increases the line indentation of selected text in the Code Editor window. If no text is selected, the indentation is applied to the current line.
Ctrl+Shift+Comma (,)	Decreases the line indentation of selected text in the Code Editor window. If no text is selected, the indentation is applied to the current line.
Ctrl+U	Changes the selection to lowercase characters.
Ctrl+V	Inserts the Clipboard contents at the current cursor location.
Ctrl+X	Cuts the selected text.
Ctrl+Y	Redoes the most recently undone action.
Ctrl+Z	Undoes the last action.
Ctrl+Minus Sign (-)	Adds and removes the double-hyphen (--) SQL comment indicator. To turn existing text into comments, select the text in the Code Editor window and press Ctrl+minus sign. The double-hyphen, SQL comment indicator is added to the start of the lines that contain the selected text. If no text is selected, the comment indicator is added to the start of the current line. To remove a comment indicator, select the text and press Ctrl +minus sign. See "Comments" [*SQL Anywhere Server - SQL Reference*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Function key	Description
Ctrl+Forward Slash (/)	Adds and removes the double-slash (//) SQL comment indicator. To turn existing text into comments, select the text in the Code Editor window and press Ctrl+forward slash. The double-slash, SQL comment indicator is added to the start of the lines that contain the selected text. If no text is selected, the comment indicator is added to the start of the current line. To remove a comment indicator, select the text and press Ctrl +forward slash. See "Comments" [*SQL Anywhere Server - SQL Reference*].
Delete	Deletes the selection.
Down Arrow	Moves the cursor down one line.
End	Moves the cursor to the end of the current line.
F3	Opens the **Find/Replace** window where you can search for and replace the specified text if you have not searched for text in the current window. Otherwise, this searches for the next occurrence of the specified text.
Home	Move the cursor to the start of the current line or to the start of the text on the current line.
Left Arrow	Moves the cursor one character to the left.
Page Down	Moves the cursor to the end of the current page.
Page Up	Moves the cursor to the top of the current page.
Right Arrow	Moves the cursor one character to the right.
Shift+Down Arrow	Extends the selection down one line.
Shift+End	Selects the current line.
Shift+F3	Opens the **Find/Replace** window where you can search for and replace the specified text if no text is selected. If text is selected, finds the previous occurrence of the selected text.

Function key	Description
Shift+F10	Displays the popup menu for the area that has focus. This keyboard shortcut is an alternative to right-clicking an area.
Shift+Home	Extends the selection to the start of the text on the current line.
Shift+Left Arrow	Extends the selection one character to the left of the currently selected character(s).
Shift+Page Down	Extends the selection down one page.
Shift+Page Up	Extends the selection up one page.
Shift+Right Arrow	Extends the selection one character to the right of the currently selected character(s).
Shift+Up Arrow	Extends the cursor up one line.
Up Arrow	Moves the cursor up one line.

Using the Log Viewer

The Log Viewer is a window in Sybase Central that displays and stores product messages. It displays the following types of messages:

- **Information** Basic information about your current session.
- **Warning** Warning messages about actions that have occurred.
- **Error** Error messages about actions that have failed.

You can filter these messages to show only a certain type or number, or choose to show only messages from a particular plug-in. You can also save messages to a file or clear all messages from the list.

When you are working in Sybase Central, you can access the Log Viewer through the **Tools** menu.

To open the Log Viewer (Sybase Central)

1. Choose **Tools** » **Log Viewer**.

   The Sybase Central Log Viewer appears, showing the current messages (if any exist).

2. Use the **View** menu to configure the types of messages that are logged.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Using the SQL Anywhere 12 plug-in

You can use the **SQL Anywhere 12** plug-in to upgrade existing databases, create new databases, and administer databases. You can choose the mode from the **Mode** menu or by clicking the toolbar button for the mode.

The **SQL Anywhere 12** plug-in can operate in any of the following modes:

- **Design mode** While working in **Design** mode, you can create and modify database objects such as tables, users, triggers, indexes, remote database servers, and so on. You can also add data to tables, create new databases, and upgrade existing databases.

  For more information about tasks you can perform on a SQL Anywhere database while in **Design** mode, see "Working with database objects" [*SQL Anywhere Server - SQL Usage*].

- **Debug mode** While working in **Debug** mode, you can use the SQL Anywhere debugger to assist you in developing SQL stored procedures, triggers, and event handlers.

  For more information about using **Debug** mode, see "Debugging procedures, functions, triggers, and events" [*SQL Anywhere Server - SQL Usage*].

- **Application Profiling mode** While working in **Application Profiling** mode, you can configure application profiling or diagnostic tracing for your database. The data that is generated helps you understand how applications interact with the database and can also help you identify and eliminate performance problems.

  For more information about using **Application Profiling** mode, see "Application profiling" [*SQL Anywhere Server - SQL Usage*].

Viewing SQL statements and utility commands generated by wizards

Most **SQL Anywhere** plug-in wizards that create a database object or run a database utility include a page at the end of the wizard. This page displays the SQL statements and utility commands that are executed when you click **Finish**.

Clicking **Finish** executes the SQL statements and/or runs the utility commands. Alternatively, you can copy the SQL statement to the clipboard, click **Cancel** to exit the wizard, and then execute the SQL statements via Interactive SQL.

This feature allows you to use the wizards to generate SQL scripts without modifying the database.

See also

- "Working with the Connect window" on page 92
- "Working with database objects" [*SQL Anywhere Server - SQL Usage*]
- "Create a Windows Mobile database using Sybase Central" on page 355

Copying database objects in the SQL Anywhere 12 plug-in

In the **SQL Anywhere 12** plug-in, you can copy existing database objects and insert them into another location in the same database or in a completely different database.

To copy an object, select the object in the left pane of Sybase Central and drag it to the appropriate folder or container, or copy the object and then paste it in the appropriate folder or container. A new object is created, and the original object's code is copied to the new object. When copying objects within the same database, you must rename the new object.

You can also paste objects onto other objects in the database. For example, if you paste a table onto a user, this gives the user permissions on the table.

In Sybase Central, when you copy any of the objects from the following list, the SQL for the object is copied to the clipboard so it can be pasted into other applications, such as Interactive SQL or a text editor.

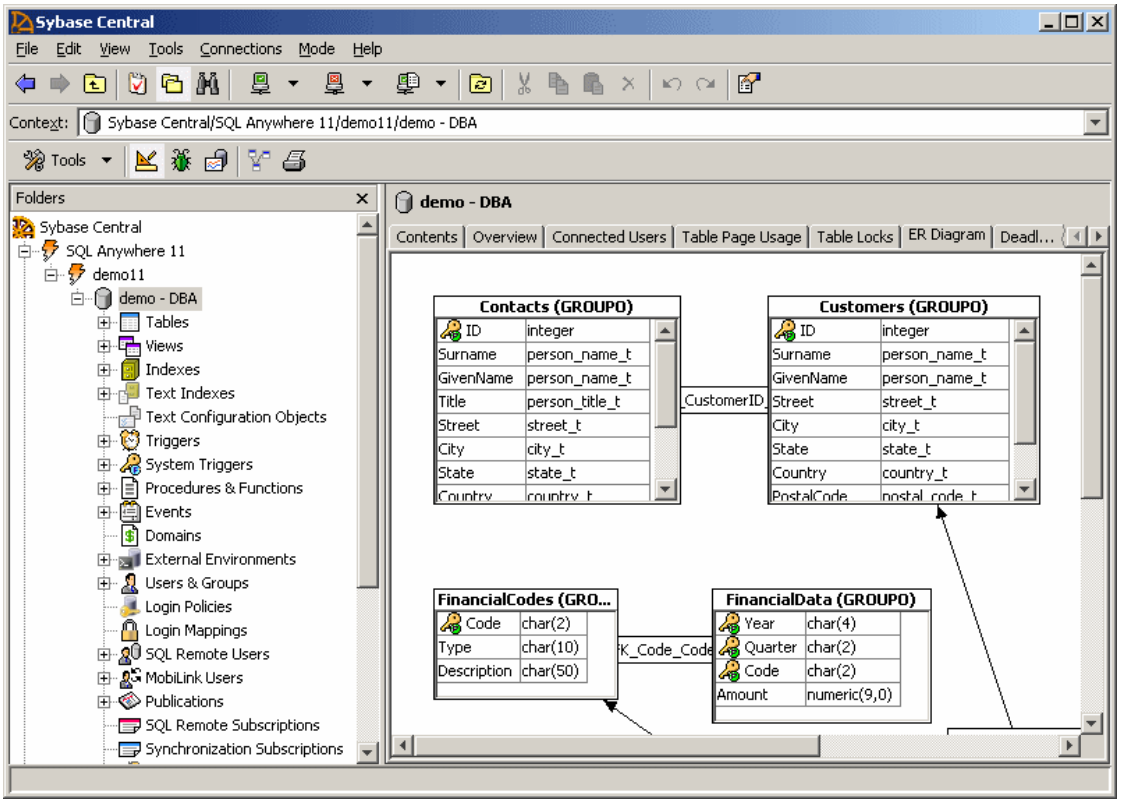
 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For example, if you copy an index in Sybase Central and paste it into a text editor, the CREATE INDEX statement for that index appears. You can copy the following objects in the **SQL Anywhere 12** plug-in:

- Articles
- Check constraints
- Columns
- Dbspaces
- Directory access servers
- Domains
- Events
- External environments
- External logins
- Foreign keys
- Indexes
- Login mappings (integrated logins and Kerberos logins)
- Login policies
- Maintenance plan reports
- Maintenance plans
- Message types
- MobiLink users
- Primary keys
- Procedures and functions
- Publications
- Remote servers
- Schedules
- Sequence Generators
- Spatial Reference Systems
- Spatial Units of Measure
- SQL Remote subscriptions
- Synchronization subscriptions
- System triggers
- Text configuration objects
- Text indexes
- Tables
- Triggers
- Unique constraints
- Users and groups
- Views
- Web services

Viewing entity-relationship diagrams from the SQL Anywhere 12 plug-in

When you are connected to a database from the **SQL Anywhere 12** plug-in, you can view an entity-relationship diagram of the tables in the database. Select the database, and then click the **ER Diagram** tab in the right pane to see the diagram.



When you rearrange objects in the diagram, the changes persist between Sybase Central sessions. Double-clicking a table takes you to the column definitions for that table.

The tables that appear in the diagram are subject to the filtering set for the database. Filtering is done by owner.

To change the tables included in the entity-relationship diagram

1. Select the database in the left pane of Sybase Central, and then choose **File** » **Configure Owner Filter**.

2. Select the database users whose tables you want to see in the entity-relationship diagram, and then click **OK**.

3. Choose **File** » **Filter Objects By Owner**.

4. Click the **ER Diagram** tab in the right pane.

5. Choose **File** » **Choose ER Diagram Tables**.

6. In the **Choose ER Diagram Tables** window, use the **Add** and **Remove** buttons to customize the tables that appear in the **Selected Tables** list.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

7. Click **OK**.

See also

- "Creating a SQL Anywhere database" on page 5

Monitoring database health and statistics

In Design mode, the **Overview** tab provides a high-level view of the database server and its features. This tab contains the following components:

- **Database** Located in the top left corner, this pane displays general information about the database server.

  To update the SQL Anywhere database server software click **Check For Updates**. See "Checking for software updates" on page 762.

- **Features** Located in the left bottom corner, this pane provides a visual representation of the database and its products and features. Clicking a node in the diagram expands the accompanying section in the **Health And Statistics** pane on the right; clicking the node again collapses the section.

  Note
  You must initiate the retrieval of MobiLink and QAnywhere information; otherwise, these nodes appear as unknowns (greyed-out). See "MobiLink, QAnywhere, And Notifiers" on page 696.

- **Health And Statistics** Located on the right, this pane displays statistics and information relating to the overall status of the database. The following collapsible panes are available:

  - **Statistics** Displays general statistics, such as the number of pages read and written to disk. It displays a warning if there are any unscheduled requests. Click the warning to learn more.

  - **Dbspaces** Displays a table listing all dbspaces. It displays a warning if a dbspace has less than 10% of free disk space remaining, or if a dbspace file cannot be found. Click the warning to learn more.

  - **Transaction Logs** Displays information for the transaction log and the transaction log mirror, if applicable. This pane appears only when the database has a transaction log. It displays a warning if a log file has less than 10% of free disk space remaining. Click the warning to learn more.

  - **Connected Users** Displays connected user and transaction statistics. Shows a table of the top 5 transaction times, if there are any. It shows a table of all blocked connections, if there are any. Displays a warning if there are any blocked connections. Click the warning to learn more.

  - **Database Mirroring** Displays information for the primary, arbiter, mirror, and copy servers and for a database mirroring or read-only scale-out system when you are connected to the primary server for the system. This pane appears only when mirroring or scale-out is being used. It displays a warning if the arbiter or mirror server is disconnected. Click the warning to learn more.

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Remote Servers** Displays a table of the remote servers used by the database. This pane appears only when a remote server exists. It displays a warning if a remote server is disconnected or a remote server cannot establish a connection. Click the warning to learn more.

  Note
  For JDBC remote servers, a warning appears that a connection has been disconnected or lost only if the JDBC remote server attempts to access a proxy object. See "Configuration notes for JDBC classes" [*SQL Anywhere Server - SQL Usage*].

- **MobiLink, QAnywhere, And Notifiers** Displays statistics for MobiLink, QAnywhere, and Notifiers. This pane only appears when MobiLink tables and views exist in the database.

  Note
  You must click the **Refresh** button to refresh the information in this pane. Unlike the information in the other panes, the information in this pane is not refreshed when you choose **View** » **Refresh** (or click the **Refresh** toolbar icon). You must refresh this information separately because refreshing could affect the database's performance.

- **SQL Remote Users** Displays a table of all SQL Remote users, and their most recent send and receive times. This pane only appears when the database has SQL Remote users.

Documenting a database

You can generate documentation about objects in a SQL Anywhere database using the **Database Documentation Wizard**. The generated documentation contains information about the following database objects:

- events
- functions
- procedures
- tables
- triggers
- views

In addition to the object definitions, the documentation also shows the dependencies and references for each object. For example, documentation for the procedure *dbo.sa_migrate_data* includes the tables that it updates, inserts into, and deletes from, and the name of the procedure that calls it. You can choose to include object comments and systems procedures in the documentation.

The generated documentation is saved to HTML files, which makes it easy to navigate and review. This documentation is useful for documenting and reviewing your system.

To generate database documentation (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database you want to generate documentation for.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

2. From the **Tools** menu, choose **SQL Anywhere 12** » **Generate Database Documentation**.
3. Follow the instructions in the **Database Documentation Wizard**.

Configuring the left pane in Sybase Central

You can specify which database objects appear in the left pane of Sybase Central.

To configure the type filter

1. Connect to a database.
2. In the left pane, right-click the database and choose **Configure Type Filter**.
3. Select the object types that you want to view in the left pane in Sybase Central.
4. (Optional) Select **Make This The Default Type Filter For New Databases** to set the default type filter to the one configured. The default type filter is used whenever you connect to a database, such as a new database, that does not have type filter specified.
5. Click **OK**.

Logging database changes

You can log to a *.sql* file all SQL statements executed by the SQL Anywhere plug-in that modify the database. See "Log SQL statements in Sybase Central" on page 42.

Using Interactive SQL

Interactive SQL is a tool included with SQL Anywhere that lets you execute SQL statements, build scripts, and display database data for both SQL Anywhere and UltraLite databases. You can use Interactive SQL for:

- Sending SQL statements to the database server. See "Executing SQL statements from Interactive SQL" on page 705.
- Browsing the information in a database. See "Executing SQL statements from Interactive SQL" on page 705.
- Editing data in result sets. See "Editing result sets in Interactive SQL" on page 715.
- Loading data into a database. See "Import data with the Import Wizard" [*SQL Anywhere Server - SQL Usage*].
- Exporting query results to a file or another database. See "Export query results" [*SQL Anywhere Server - SQL Usage*].

- Running script files. See "Run SQL command files in Interactive SQL" [*SQL Anywhere Server - SQL Usage*].
- Running the Index Consultant, a tool that helps you improve query performance. See "Index Consultant" [*SQL Anywhere Server - SQL Usage*].
- Accessing the Query Editor, a tool that helps you design, analyze, and test all kinds of queries. See "Using the Query Editor" on page 720.
- Viewing execution plans for a query. See "Using the Plan Viewer to view graphical plans" on page 723.
- Viewing images, including SVGs. See "Viewing images in Interactive SQL" on page 725.
- Viewing spatial data. See "View spatial data as images" [*SQL Anywhere Server - Spatial Data Support*].

Interactive SQL is available on Windows, Solaris, Linux, and Mac OS X, see http://www.sybase.com/detail?id=1061806.

Mac OS X note
The administration tools only run on Intel-based Macintosh computers with 64-bit processors supported by the Apple JDK 1.6 (Mac OS X 10.5.2 or later). See http://www.sybase.com/detail?id=1061806.

SQL statements used only from Interactive SQL

Interactive SQL supports all SQL statements supported by SQL Anywhere and UltraLite databases, and several SQL statements that can be used only from Interactive SQL. See "Interactive SQL SQL statements" on page 733.

Starting Interactive SQL

There are several ways you can start Interactive SQL: from a command prompt, from the Windows **Start** menu, and from within Sybase Central.

To start Interactive SQL (command line)

- Run the following command:

  ```
  dbisql
  ```

  If you do not include the -c option, which specifies the connection parameters for the database, or if you supply insufficient connection parameters, the **Connect** window appears, where you can enter connection information for the database. See "Connection parameters" on page 265.

  To start Interactive SQL and connect to the sample database, run the following command:

  ```
  dbisql -c "UID=DBA;PWD=sql;DSN=SQL Anywhere 12 Demo"
  ```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For information about the supported options, see "Interactive SQL utility (dbisql)" on page 812.

To start Interactive SQL (Windows)

1. From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Interactive SQL**.
2. Enter the connection information for your database in the **Connect** window.

   For example, to connect to the SQL Anywhere sample database:

   a. In the **Authentication** field, choose **Database**.

   b. From the **Action** dropdown menu list choose **Connect With An ODBC Data Source**.

   c. Click **ODBC Data Source Name**, and then in the box below type **SQL Anywhere 12 Demo**.
3. Click **Connect**.

To start Interactive SQL (Sybase Central)

1. From the **Tools** menu, choose **SQL Anywhere 12** » **Open Interactive SQL**.
2. Enter the connection information for your database in the **Connect** window.

   For example, to connect to the SQL Anywhere sample database:

   a. In the **Authentication** field, choose **Database**.

   b. From the **Action** dropdown menu list choose **Connect With An ODBC Data Source**.

   c. Click **ODBC Data Source Name**, and then in the box below type **SQL Anywhere 12 Demo**.
3. Click **Connect**.

Tip

You can also use one of the following methods to access Interactive SQL from Sybase Central:

- Selecting a database, and choosing **Open Interactive SQL** from the **File** menu.
- Right-clicking a database, and choosing **Open Interactive SQL**.
- Right-clicking a stored procedure, and choosing **Execute From Interactive SQL**. Interactive SQL opens with a CALL to the procedure in the **SQL Statements** pane and executes the stored procedure.
- Right-clicking a table or view and choosing **View Data In Interactive SQL**. Interactive SQL opens with a `SELECT * FROM table-name` and executes the query.

Starting Interactive SQL on Unix

Note

The following steps assume that you have already sourced the SQL Anywhere utilities. See "Setting environment variables on Unix and Mac OS X" on page 377.

To start Interactive SQL (Unix command line)

1. In a terminal session, run the following command:

   ```
   dbisql
   ```

2. Enter the connection information for your database in the **Connect** window.

3. Click **Connect**.

To start Interactive SQL (Mac OS X)

1. In the Finder, double-click **Interactive SQL** in */Applications/SQLAnywhere12*.

2. Enter the connection information for your database in the **Connect** window.

3. Click **Connect**.

The following steps can be used if you are using a version of Linux that supports the Linux desktop icons and if you chose to install them when you installed SQL Anywhere.

To start Interactive SQL (Linux desktop icons)

1. From the **Applications** menu, choose **SQL Anywhere 12** » **Administration Tools** » **Interactive SQL**.

2. Enter the connection information for your database in the **Connect** window.

3. Click **Connect**.

See also

- "Working with the Connect window" on page 92

Navigating Interactive SQL

The Interactive SQL window is divided into panes:

- **SQL Statements** This pane provides a place for you to type SQL statements to access and modify your data.

  The **SQL Statements** pane contains a column on the left that shows line numbers. These line numbers allow you to do the following:

  - **Select a line** Click a line number to select a line. Alternatively place your cursor in the line, and press Ctrl+comma(,).

  - **Select multiple lines** Click and drag to select multiple lines.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Select a statement** Double-click a line to select the entire SQL statement that corresponds to the line. Alternatively, place your cursor in the statement, and press Ctrl+period(.).

  See "Interactive SQL keyboard shortcuts" on page 734.

- **Results** The **Results** pane has two tabs: **Results** and **Messages**. The tabs appear at the bottom of the **Results** pane.

  The **Results** tab displays the results of commands that you execute. For example, if you use SQL statements to search for specific data in the database, the **Results** tab displays the columns and rows that match the search criteria in the pane above. You can edit the result set on the **Results** tab. See "Editing result sets in Interactive SQL" on page 715.

  The **Messages** tab displays messages from the database server about the SQL statements that you execute in Interactive SQL.

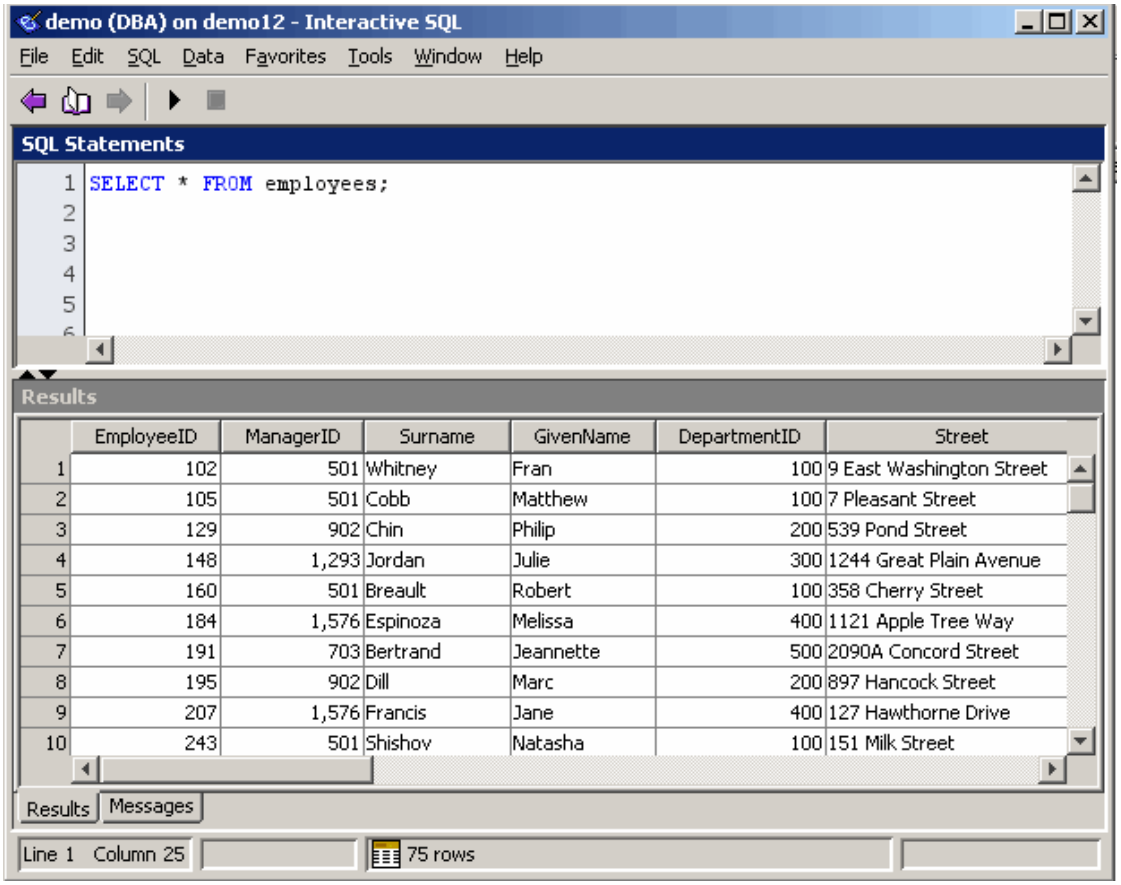
  Results of graphical plans for SQL Anywhere databases and text plans for UltraLite databases are displayed in separate Plan Viewer window(s). See "To create a graphical plan with detailed and node statistics" on page 723.

When you are connected to a database from Interactive SQL, the title bar displays connection information, as follows:

```
database-name ( userid ) on server-name
```

For example, if you connect to the sample database using the SQL Anywhere 12 Demo ODBC data source, the title bar contains the following information:

```
demo ( DBA ) on demo12
```



Interactive SQL windows

You can access all the windows in Interactive SQL through the **Tools**, **Data**, and **Favorites** menus. With these windows, you can configure Interactive SQL settings, search for table and procedure names to insert into your queries, edit your queries, export a result set, and save files and connection information as favorites.

The **Tools** menu contains the following windows:

- **Lookup Table Name** The **Lookup Table Name** window lets you browse table and column names and insert them into the **SQL Statements** pane. See "Looking up tables, columns, and procedures" on page 714.
- **Lookup Procedure Name** The **Lookup Procedure Name** window lets you browse procedure names and insert them into the **SQL Statements** pane. See "Looking up tables, columns, and procedures" on page 714.
- **Edit Query** The Query Editor provides a graphical way to create and edit SELECT statements in Interactive SQL. See "Using the Query Editor" on page 720.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Index Consultant** The Index Consultant guides you in the proper selection of indexes. You can use the Index Consultant to analyze the benefits of indexes for an individual query. See "Obtain Index Consultant recommendations for a query" [*SQL Anywhere Server - SQL Usage*].

- **Plan Viewer** The **Plan Viewer** is a graphical tool for viewing graphical plans for SQL Anywhere databases and text plans for UltraLite databases. See "Using the Plan Viewer to view graphical plans" on page 723.

- **Spatial Viewer** The **Spatial Viewer** is a graphical tool for viewing spatial data. See "View spatial data as images" [*SQL Anywhere Server - Spatial Data Support*].

- **Options** The **Options** window sets options for commands, appearance, importing and exporting data, and messages in Interactive SQL.

The **Data** menu contains the following windows:

- **Export** Opens the **Export Wizard**, which allows you to export a result set. See "Export data with the Export Wizard" [*SQL Anywhere Server - SQL Usage*].

- **Import** Opens the **Import Wizard**, which allows you to import data from a file or database. See "Import data with the Import Wizard" [*SQL Anywhere Server - SQL Usage*].

The **Favorites** menu contains the following windows:

- **Add To Favorites** This window allows you to save SQL files, SQL statements, and connection information as favorites. See "Using favorites" on page 711.

- **Organize Favorites** This window allows you to maintain and organize your favorites. See "Using favorites" on page 711.

- **Import Favorites** This window allows you to import favorites from a *.fav* file. See "Sharing Favorites" on page 712.

- **Export Favorites** This window allows you to export favorites to a *.fav* file. See "Sharing Favorites" on page 712.

- **Show Favorites** Opens the **Favorites** window on the left side of the Interactive SQL window. See "Using favorites" on page 711.

Customizing Interactive SQL

You can configure settings for the tabs and panes in Interactive SQL using the **Options** window. When you deploy Interactive SQL, you can control which features are shown or enabled in the **Options** window. See "Configuring the administration tools" [*SQL Anywhere Server - Programming*].

To customize Interactive SQL

1. In Interactive SQL, choose **Tools** » **Options**.

2. In the left pane, click an option and specify the options that you want.

3. Click **OK**.

To configure how result sets are displayed in Interactive SQL

1. Choose to show the results as a scrollable table or as text.

   For example, choose **Data** and then choose one of the following:

   - **Show Results As Scrollable Table** You can edit the result set in this format. This is the default.
   - **Show Results As Text** This option displays the result set as text using a monospaced font. The result set is not editable in this format.

2. Execute a query.

   You must execute a new query for the changes to table editing to take effect.

3. Right-click within the result set and choose one of the following:
   - **Size Columns To Data** The table columns are made wide enough so that the values they contain are not truncated.
   - **Size Columns To Window** The table columns are made the same width so they all fit within the window without a horizontal scroll bar.

Disable warning messages

You can disable some of the warning messages that appear in Interactive SQL. For example you can suppress the warning that appears when you have unsaved text in the **SQL Statements** pane and you choose **File** » **Exit**.

To disable warning messages

1. Choose **Tools** » **Options** » **Messages**.

2. Clear the messages listed in the **Optional Messages** list.

See also

- "Troubleshooting unexpected symbols when viewing data" on page 411

Working with files in Interactive SQL

Setting Interactive SQL as the default editor for *.sql* files

On Windows platforms you can make Interactive SQL the default editor for *.sql* command files.

To make Interactive SQL the default editor for .sql files

1. From Interactive SQL, choose **Tools** » **Options**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

2. In the left pane, click **General**.
3. Click **Make Interactive SQL The Default Editor For .SQL Files And Plan Files**.
4. Click **OK**.

For more information about using Interactive SQL with command files, see:

- "Using SQL command files" [*SQL Anywhere Server - SQL Usage*]
- "Run SQL command files in Interactive SQL" [*SQL Anywhere Server - SQL Usage*]

Backing up *.sql* files

When you edit a *.sql* file, Interactive SQL automatically creates a backup copy after:

- You execute a statement.
- The file has remained unchanged for 30 seconds.

If you are editing a file and Interactive SQL closes unexpectedly, you are prompted to recover the file, the next time you start Interactive SQL.

You can also back up your favorites. See "Sharing Favorites" on page 712.

Executing SQL statements from Interactive SQL

One of the primary uses of Interactive SQL is to browse table data. Interactive SQL retrieves information by sending a request to your database server. The database server, in turn, looks up the information, and returns it to Interactive SQL.

After you execute a SELECT statement, the result set appears on the **Results** tab in the **Results** pane. By default, row numbers appear to the left of the result set.

Note
The database server creates schema locks on tables that you view in Interactive SQL, even if you do not modify the table.

However, you can configure Interactive SQL to attempt to release the database schema locks it creates when it displays your result set. To do so, in Interactive SQL, choose **Tools** » **Options** » **SQL Anywhere**, and select **Automatically Release Database Locks**.

When this option is selected, after you execute a statement that returns a result set, Interactive SQL checks if your connection has any uncommitted changes in the database. If none exist, then Interactive SQL releases your schema locks; otherwise, Interactive SQL does not release your schema locks. That is, Interactive SQL does not release your schema locks if you have any uncommitted changes to the database.

To execute all SQL statements

1. Type your query in the **SQL Statements** pane.
2. Press F5, or choose **SQL** » **Execute** to execute the statement.

To execute selected SQL statements

1. Type your queries in the **SQL Statements** pane and select the query.
2. Press F9, or choose **SQL** » **Execute Selection** to execute the statement.

To execute SQL statements individually, for example when debugging, you can use **Single Step** from the **SQL** menu. **Single Step** executes a specified statement and then selects the next statement to be executed. To execute the next statement, run **Single Step** again.

To execute SQL statements one at a time

1. Type your queries in the **SQL Statements** pane.
2. Place your cursor in the statement that you want to execute.
3. From the **SQL** menu, choose **Single Step** or press Shift+F9 to execute the specified statement.

   When the SQL statement executes, the next SQL statement is selected.
4. To execute the selected SQL statement, press Shift+F9.
5. Repeat the previous step until there are no more selected statements to execute.

Configuring the Execute Statements toolbar button

You can also click the **Execute Statements** button to execute the statements in the **SQL Statements** pane. This button can be set to execute all SQL statements or only execute the selected statements.

To configure the Execute Statements toolbar button

1. From the **Tools** menu, choose **Options**.
2. Click **Toolbar**:

   To execute all SQL statements, select **Execute All Statement(s)**. This is the default setting.

   To execute only the selected SQL statements, select **Execute Selected Statement(s)**.

See also

- "Configuring the administration tools" [*SQL Anywhere Server - Programming*]

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Inserting comments

Comments are used to attach explanatory text to SQL statements or statement blocks. The database server does not execute comments. SQL Anywhere supports the following types of comments: -- (double hyphen), // (double slash), and /* ... */ (slash-asterisk). See "Comments" [*SQL Anywhere Server - SQL Reference*].

To add or remove comment indicators

- To turn existing text into comments, select the text in the **SQL Statements** pane and press Ctrl+Minus Sign (-) to add double hyphen comment indicators or Ctrl+Forward Slash (/) to add double slash comment indicators. The SQL comment indicator is added to the beginning of each line with selected text.

  If no text is selected, the comment indicator is added to the beginning of the current line.

  To remove a comment indicator, select the text and press Ctrl+Minus Sign (-) to remove double hyphen comment indicators or Ctrl+Forward Slash (/) to remove double slash comment indicators.

Indenting SQL statements

To change the indentation of SQL statements

1. Select the text in the **SQL Statements** pane that you want to indent. If no text is selected, the indentation is applied to the current line.
2. Press Ctrl+Period (.) to add or increase indentation.

   Press Ctrl+Comma (,) to remove or decrease indentation.

To change the number of spaces that are indented

1. From the **Tools** menu, choose **Options** » **Editor** » **Tabs**.
2. Type a new number in the **Indent Size** field.

Executing multiple SQL statements

You can execute multiple SQL statements from Interactive SQL as long as each statement ends with a command delimiter. The command delimiter is set with the command_delimiter option, and is a semicolon (;) by default. An alternative to using the semicolon is to enter the separator **go** on a line by itself, at the beginning of the line. See "command_delimiter option [Interactive SQL]" on page 742.

Tip
You can press F9 to execute only the selected text in the **SQL Statements** pane.

You can press Shift+F9 to execute only the selected statement in the **SQL Statements** pane and select the next statement for execution.

Showing multiple result sets

By default, Interactive SQL shows the first result set of the most-recently executed statement. To enable Interactive SQL to return multiple result sets, see "Returning multiple result sets from procedures" [*SQL Anywhere Server - SQL Usage*].

See also

- "Troubleshooting unexpected symbols when viewing data" on page 411

Executing command files

Command files are text files that contain SQL statements, and are useful if you want to run the same SQL statements repeatedly. You can use Interactive SQL to open, view, run, and save command files.

You can execute command files in any of the following ways from Interactive SQL:

- You can use the Interactive SQL READ statement to execute command files. For example, the following statement executes the file *temp.sql*:

  ```
  READ temp.sql;
  ```

- You can load a command file into the **SQL Statements** pane and execute it directly from there.

  You load command files into the **SQL Statements** pane by choosing **File** » **Open**. Enter the file name, for example *temp.sql*, when prompted.

- You can run a command file without loading it by choosing **File** » **Run Script**.

- You can supply a command file as a command line argument for Interactive SQL.

See "Run SQL command files in Interactive SQL" [*SQL Anywhere Server - SQL Usage*].

Executing COMMIT and ROLLBACK statements in Interactive SQL

You can execute a COMMIT statement by:

- Choosing **SQL** » **Commit**.
- Pressing Ctrl+Shift+C.
- Typing **Commit** into the **SQL Statements pane**, and then executing the statement.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can execute a ROLLBACK statement by:

- Choosing **SQL** » **Rollback**.
- Pressing Ctrl+Shift+R.
- Typing **Rollback** into the **SQL Statements pane**, and then executing the statement.

Note
Executing a COMMIT or ROLLBACK via the **SQL** menu or a keyboard shortcut does not modify the contents of the **SQL Statements** pane; however, the **Results** tab in the **Results** pane is cleared.

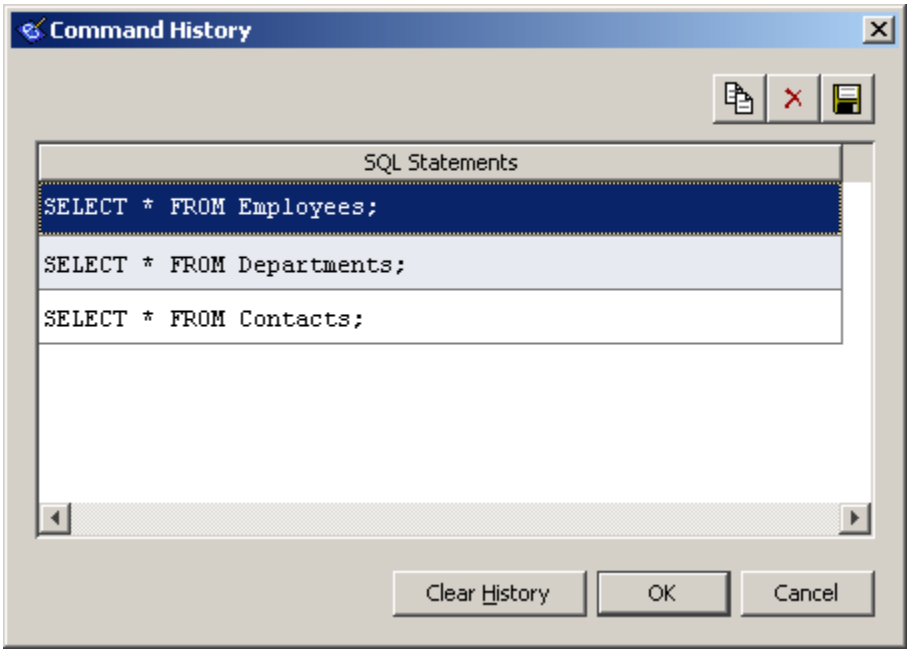
See also

- "auto_commit option [Interactive SQL]" on page 740
- "commit_on_exit option [Interactive SQL]" on page 743
- "COMMIT statement" [*SQL Anywhere Server - SQL Reference*]
- "ROLLBACK statement" [*SQL Anywhere Server - SQL Reference*]
- "Interactive SQL options" on page 739

Recalling commands

When you execute a command, Interactive SQL automatically saves it in a history list that persists between Interactive SQL sessions. Interactive SQL maintains a record of up to 50 of the most recent commands.

You can view the entire list of commands in the **Command History** window. To access the **Command History** window, press Ctrl+H, or click the **Open A List Of Past SQL Statements** button on the toolbar.



The most recent commands appear at the bottom of the list. To recall a command, select it and then click **OK**. It appears in the **SQL Statements** pane of Interactive SQL. You can select multiple commands from the **Command History** window.

You can also recall commands without the **Command History** window. Use the **Recall Previous SQL Statement** and **Recall Next SQL Statement** icons in the toolbar to scroll back and forward through your commands, or press Alt+Right Arrow and Alt+Left Arrow, respectively.

Note

If you execute a SQL statement that contains password information (CREATE USER, GRANT REMOTE DBA, CONNECT, or CREATE EXTERNLOGIN), the password information appears in the **Command History** window for the duration of the current Interactive SQL session.

When the command history is viewed in subsequent Interactive SQL sessions, passwords are replaced with ... in any of these statements that contain password information. For example, if you execute the following statement in Interactive SQL:

```
CREATE USER testuser
 IDENTIFIED BY testpassword;
```

the following statement appears in the **Command History** window in subsequent Interactive SQL sessions:

```
CREATE USER testuser
 IDENTIFIED BY ...;
```

Copying commands from the Command History window

You can copy commands from the **Command History** window to use elsewhere. When you copy multiple commands, they are separated by the command delimiter (a semicolon by default).

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To copy commands from the Command History window

1. Open the **Command History** window.
2. Select the command or commands, and then press Ctrl+C or click **Copy**.
3. Click **OK** to copy the selected statements to the **SQL Statements** pane of Interactive SQL.

Saving commands from the Command History window

You can also save commands in text files so that you can use them in a subsequent Interactive SQL session.

To save the command history to a file

1. Open the **Command History** window.
2. Click the **Save History As .SQL File** button or press Ctrl+S.
3. In the **Save As** window, specify a location and name for the file.

   The command history file has a *.sql* extension.
4. Click **Save** when finished.

You can also save SQL command files in a **Favorites** list. See "Using favorites" on page 711.

Removing commands from the Command History window

The contents of the **Command History** window persist between Interactive SQL sessions. You can remove commands from the history in one of two ways:

- Select one or more commands and click the **Delete** button or press the Delete key to remove the selected command(s) from the window. This action cannot be undone.
- Remove all the commands from the window by clicking **Clear History**. This action cannot be undone.

Using favorites

In Interactive SQL, you can store frequently-used SQL command files and connections in a favorites list. A favorites list is specific to a single user and cannot be seen by other users.

To add a *.sql* file, SQL statements, or Connection to favorites

1. Open the SQL command file that you want to add to your favorites.
2. From the **Favorites** menu, choose **Add To Favorites**.
3. Select **Add the open file '*filename*'**. In the **Name** field, type a name for the *.sql* file.
4. Click **OK**.

To add SQL statements to favorites

1. Type the SQL commands that you want to add to your favorites in the **SQL Statements** pane. The contents in the SQL Statements pane cannot exceed 16384 characters.
2. From the **Favorites** menu, choose **Add To Favorites**.
3. Select **Add SQL Statements**. In the **Name** field, type a name for the favorite.
4. Click **OK**.

To add a connection to favorites

1. Connect to a database.
2. From the **Favorites** menu, choose **Add To Favorites**.
3. Select **Save The Connection Password**. In the **Name** field, type a name for the connection.
4. Click **OK**.

To display favorites in a sidebar

- From the **Favorites** menu, choose **Show Favorites**.

  The **Favorites** pane appears on the left side of the Interactive SQL window.

To open a favorite

- From the **Favorites** menu, choose the favorite you want to open.

To edit a favorite

1. From the **Favorites** menu, choose **Show Favorites**.

   The **Favorites** pane appears on the left side of the Interactive SQL window.
2. Select a favorite, right-click, and then choose Edit.
3. Follow the instructions in the window.
4. Click **Save**.

See also

- "Configuring the administration tools" [*SQL Anywhere Server - Programming*]

Sharing Favorites

You can export all of your favorites to a *.fav* file. And then, you can import the favorites on to another computer or save the file as a backup.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To export favorites (Interactive SQL)

1. Choose **Favorites** » **Export Favorites**.
2. Specify a file name for the *.fav* file and then click **Export**.

To import favorites (Interactive SQL)

1. Choose **Favorites** » **Import Favorites**.
2. Browse to the *filename.fav* file and then click **Import**.

Logging commands

With the Interactive SQL logging feature, you can record commands as you execute them. Interactive SQL continues to record commands until you stop the logging process, or until you end the current session. The recorded commands are stored in a log file so you can use the commands again.

Once you start logging, all commands that you try to execute are logged, including ones that do not execute properly.

To start logging commands (Interactive SQL)

1. From the **SQL** menu, choose **Start Logging**.
2. In the **Save As** window, specify a location and name for the log file. For example, name the file *mylogs.sql*.
3. Click **Save** when finished.

To start logging commands (SQL)

- Execute a START LOGGING statement.

  For example, to start logging to a file named *c:\mylogs.sql*, run the following:

  ```
  START LOGGING 'c:\mylogs.sql';
  ```

  See "START LOGGING statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*].

To stop logging commands (Interactive SQL)

- From the **SQL** menu, choose **Stop Logging**.

To stop logging commands (SQL)

- Run the following:

```
STOP LOGGING;
```

See "STOP LOGGING statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*].

Tips

- When you work with Sybase Central, you can also log commands that you execute. See "Log SQL statements in Sybase Central" on page 42.
- In addition, execution times can be included in the log file when logging and execution time reporting are both enabled. See "isql_command_timing option [Interactive SQL]" on page 747.

Canceling commands in Interactive SQL

A cancel operation stops the current processing and prompts for the next command. The **Stop** button on the Interactive SQL toolbar cancels a command.

If a command file was being processed, you are prompted for an action to take (**Stop Command File**, **Continue**, or **Exit Interactive SQL**). These actions can be controlled with the Interactive SQL on_error option. See "on_error option [Interactive SQL]" on page 752.

Looking up tables, columns, and procedures

While you are entering commands in Interactive SQL, you can look up the names of tables, columns, or procedures stored in the current database and insert them at your cursor position.

To look up the names of tables in the database

1. From the **Tools** menu, choose **Lookup Table Name** or press F7.
2. Find and select the table.
3. Click **OK** to insert the table name into the **SQL Statements** pane at the current cursor position.

To look up column names in the database

1. From the **Tools** menu, choose **Lookup Table Name** or press F7.
2. Find and select the table containing the column.
3. Click **Show Columns**.
4. Select the column and click **OK** to insert the column name into the **SQL Statements** pane at the current cursor position.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To look up the names of procedures in the database

1. From the **Tools** menu, choose **Lookup Procedure Name** or press F8.
2. Find and select the procedure.
3. Click **OK** to insert the procedure name into the **SQL Statements** pane at the current cursor position.

In the **Lookup Table Name** and **Lookup Procedure Name** windows, you can enter the first few characters of the table or procedure you are looking for. The list is narrowed to include only those items that start with the text you entered.

You can use the SQL wildcard characters '%' (percent sign) and '_' (underscore) to help narrow your search. '%' matches any string of zero or more characters, while '_' matches any one character.

For example, to list all the tables that contain the word profile, type **%profile%**.

If you want to search for a percent sign or underscore within a table name, you must prefix the percent sign or underscore with an escape character. The escape character for the SQL Anywhere ODBC driver and SQL Anywhere JDBC driver is '~' (tilde).

Tip
Interactive SQL supports text completion for database object names when you type in the **SQL Statements** pane, which can be used as an alternative to looking up table and procedure names. See "Using text completion" on page 755.

Generating SQL statements from result sets

You can create INSERT, DELETE, and UPDATE statements for selected rows in the result set.

To generate SQL statements from an Interactive SQL result set

1. Select the row(s) you want to generate a statement for.
2. Right-click the selection, and choose **Generate**, and then choose **INSERT Statement**, **DELETE Statement**, or **UPDATE Statement**.

   The statement is copied to the clipboard.

Editing result sets in Interactive SQL

Once you execute a query in Interactive SQL, you can sort and edit the result set to modify the database. You can also select rows from the result set and copy them for use in other applications. The field delimiter, quoting character, and escape character for the results are controlled by the isql_field_separator, isql_quote, and isql_escape_character options, respectively. These options can be viewed and changed in **Options** window in Interactive SQL or by executing the SET OPTION statement. See "Interactive SQL options" on page 739.

Interactive SQL supports editing, inserting, and deleting rows. Editing the result set has the same effect as executing UPDATE, INSERT, and DELETE statements. After editing a result set, the equivalent INSERT, UPDATE, and DELETE statements are added to the Interactive SQL command history. See "Recalling commands" on page 709.

To edit a row or value in the result set, you must have the proper permissions on the table or column you want to modify values from. For example, if you want to delete a row, then you must have DELETE permission for the table the row belongs to.

You cannot edit a result set if you:

- Select columns from a table with a primary key, but do not select all the primary key columns.
- Attempt to edit the result set of a JOIN (for example, if there is data from more than one table in the result set).
- Attempt to edit a table that has its editing disabled, see "Enabling and disabling table editing" on page 717.

Editing the result set may fail if you:

- Attempt to edit a row or column you do not have permission on.
- Enter an invalid value (for example, a string in a numeric column or a NULL in a column that does not allow NULLs).

When editing fails, an Interactive SQL error message appears explaining the error, and the database table values remain unchanged.

Editing table values from the Interactive SQL result set

From Interactive SQL you can change any or all the values within existing rows in database tables if you:

have UPDATE permission on the columns being modified. In addition, for SQL Anywhere and UltraLite databases, table editing must not be disabled.

When you edit the result set, you can only make changes to the values in one row at a time.

To edit a row in the result set

1. Execute a query in Interactive SQL.
2. On the **Results** tab, click the value you want to change.
3. Right-click the value and choose **Edit Row**, or press F2.

   A **...** button appears in the table cell containing the value.
4. Click **...** and choose one of the following options:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Edit In Window** Opens a window in which you can type in a new value (applies only to character data fields).
- **Set To NULL** Sets the cell value to NULL. If the column is not nullable, this menu item does not appear.
- **Set To DEFAULT** Sets the cell value to DEFAULT. This menu item appears when adding a new row to a table, and only if the column has defined a default value.
- **Load From File** Opens a file browser for you to enter a file name, then loads the contents of the cell from the file.

5. Enter the new value. If you want to change other values in the row, press Tab or Shift+Tab to move to the other values.
6. Press Enter to update the database once you are done editing values in the row.

   You can press the Esc key to cancel the change that was made to the selected value.
7. Execute a COMMIT statement to make your changes to the table permanent. For example press CTRL +SHIFT+C.

Enabling and disabling table editing

You can disable table editing via the **Options** window in Interactive SQL.

To disable table editing (Interactive SQL)

1. From the **Tools** menu, choose **Options**, and then choose **SQL Anywhere** or **UltraLite**.
2. Ensure that **Scrollable Table** is selected and select **Disable Editing**.
3. Click **OK**.
4. Execute a query.

   You must execute a new query for the changes to table editing to take effect.

When deploying Interactive SQL, you can also disable table editing. See "Configuring the administration tools" [*SQL Anywhere Server - Programming*].

Inserting rows into the database from the Interactive SQL result set

Interactive SQL allows you to add new rows to a table. You tab between columns in the result set to add values to the row. You must have INSERT permission on the table to add new rows.

To insert a new row into the result set

1. Right-click the result set and choose **Add Row**.

   A new blank row appears with a blinking cursor in the first value in the row.

2. Enter the new value and then press Tab to move to the next column.

   You cannot enter invalid data types into a column. For example, you cannot enter a string into a column that accepts the INT data type.

   Repeat this step until all the column values are added.

3. Press Enter to update the database.

Inserting values into columns with default values

When adding a value in a column that has a default value, the cell editor contains a list with a **(DEFAULT)** item. Select **(DEFAULT)** if you want to insert the default value. Similarly, if a column accepts NULL values, **(NULL)** appears in the list. If a column cannot be NULL and does not have a default value, you must enter a value.

Inserting values into computed columns

If the result set contains a computed column and you do not specify a value for the computed column, the value is calculated when the database is updated. However, if you specify a value for the computed column, the database is updated with the specified value, and a value is not calculated for the computed column.

Inserting new rows using the INPUT statement

An alternative to inserting new rows from the result set in Interactive SQL is to add rows using the INPUT statement with the PROMPT clause. When the PROMPT clause is specified, Interactive SQL prompts you for the value for each column in the table. For example, to add a new row to the Products table and be prompted for the values for each column, you would execute the following statement in Interactive SQL:

```
INPUT INTO Products PROMPT;
```

See also

- "INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "input_format option [Interactive SQL]" on page 745

Deleting rows from the database using Interactive SQL

You can also delete rows from a database table in Interactive SQL. You must have DELETE permission on the table to delete rows.

To delete a row from the result set

1. Select the row(s) you want to delete. To select a row(s):
   - Press and hold the Shift key while clicking the row(s).
   - Press and hold the Shift key while using the Up or Down Arrow.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

If you want to delete non-consecutive rows, you must delete each row individually.

2. Press Delete.

   The selected row(s) are removed from the database table.

3. Execute a COMMIT to make the change permanent.

Copying columns, rows, and cells from an Interactive SQL result set

You can copy cells, rows, and columns directly from the result set in Interactive SQL and then paste them into other applications. Copying rows and columns copies both the column headings and table data into the clipboard. You can only copy one column at a time.

Copied data is formatted according to the following Interactive SQL options:

- "isql_field_separator option [Interactive SQL]" on page 748
- "isql_escape_character option [Interactive SQL]" on page 747
- "isql_quote option [Interactive SQL]" on page 750

You can also change these options from the Interactive SQL **Options** menu, by choosing **Import/Export**.

If these options are set to their defaults, the copied data is comma-delimited and the strings are enclosed in single quotes.

To copy rows from the Interactive SQL result set

1. To copy one row, right-click any cell in the row and choose **Copy Data** » **Rows**.

   To copy multiple rows, hold the Ctrl key while clicking cells in the rows, and then right-click and choose **Copy Data** » **Rows**.

   The selected row(s), including their column headings, are copied to the clipboard.

2. You can now paste the row(s) into other applications.

To copy column(s) from the Interactive SQL result set

1. To copy one column, right-click any cell in the column and choose **Copy Data** » **Columns**.

   To copy multiple columns, hold the Ctrl key while clicking cells in the columns, and then right-click and choose **Copy Data** » **Columns**.

   If the **Results** pane does not contain the entire result set, you are prompted to fetch the remaining results before selecting them. Otherwise, only those results that have been fetched so far are selected.

   The column, including the column heading, is copied to the clipboard.

2. You can now paste the column(s) into other applications.

To copy cells from the Interactive SQL result set

1. Right-click the cell you want to copy and choose **Copy Data** » **Cells**. To copy multiple cells, hold the Ctrl key while clicking cells, and then right-click and choose **Copy Data** » **Cells**.

   When you do this, no column headings are copied—only the data is copied to the clipboard, and no quoting is done.

2. You can now paste the contents of the cell(s) into other applications.

Sorting columns in an Interactive SQL result set

To sort columns in the result set

- Click a column-header in the **Results** tab, to sort the results by that column.

  When the **Results** tab does not contain the entire result set, you are prompted to fetch the remaining results. Otherwise, only the currently fetched results are sorted.

Using the Query Editor

The Query Editor is a tool in Interactive SQL that helps you build SELECT statements. You can create SQL queries in the Query Editor, or you can import queries and edit them. When you have finished your query, click **OK** to export it back into Sybase Central or Interactive SQL for processing.

To create a query using the Query Editor

1. Connect to a database from Interactive SQL.

2. Open the Query Editor.

   From the **Tools** menu, choose **Edit Query**.

   If you have a SQL statement selected in Interactive SQL, the selected statement is automatically imported into the Query Editor.

3. Create your query.

   At any time while using the Query Editor, you can click **SQL** at the bottom of the window to see the SQL statement for the query you are building. You can directly edit the code, and the fields are automatically updated in the Query Editor user interface.

4. Click **OK** to write the query to the Interactive SQL **SQL Statements** pane.

You do not need to use SQL statements to create queries with the Query Editor. However, you can use SQL statements with the Query Editor in the following ways:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- You can create a query in the **SQL Statements** pane in Interactive SQL, and import it into the Query Editor by highlighting the code before you open the editor.
- At any time while using the Query Editor, you can click **SQL** at the bottom of the window to see the SQL statements for the query you are building. You can directly edit the code, and the fields are automatically updated in the Query Editor.

  You can configure the Query Editor from Interactive SQL or Sybase Central so that the SQL is fully formed, meaning that all table and column names fully qualified and names are quoted. This extra formatting is not normally necessary, but it ensures that the SQL works in all situations. You can also choose to get a list of tables on startup.

  To configure the Query Editor

  - From the **Tools** menu, choose **Options** » **SQL Anywhere**, and then click the **Query Editor** tab.

The Query Editor provides a series of tabs that guide you through the components of a SQL query, most of which are optional. The tabs are presented in the order that SQL queries are usually built:

Tab	Description
Tables tab	Use this tab to specify the tables in your query.
Joins tab	Use this tab to specify a join strategy for combining the data in the tables. If you include more than one table in your query, you should specify a join strategy for combining the data in the tables. If you do not specify a join strategy for tables you added in the **Tables** tab, the Query Editor suggests one; if there is a foreign key relationship between the tables, it generates a join condition based on that relationship, or it suggests a cross product. When you open queries, the Query Editor accepts exactly the join strategy that you specified (and an unspecified JOIN is not defaulted to KEY JOIN, as it would be otherwise in SQL Anywhere).
Columns tab	Use this tab to specify the columns in your result set. If you do not specify columns, all columns appear.
INTO tab	Use this tab to assign results to variables.
WHERE tab	Use this tab to specify conditions for restricting the rows in your result set.
GROUP BY tab	Use this tab to group rows in the result set.
HAVING tab	Use this tab to restrict the rows in your result set based on group values.
ORDER BY tab	Use this tab to sort the rows.

The Query Editor also contains the following tools:

Window	Description
Expression Editor	Use the Expression Editor to build search conditions or define computed columns.
Derived Table	Use this window, which is nearly identical to the main Query Editor, to create derived tables and subqueries.

Each component of the Query Editor has context-sensitive online help that describes how to use the tab, and provides links into the SQL Anywhere documentation that explain relevant concepts and usage.

Query Editor limitations

The Query Editor builds SQL Anywhere SELECT statements. It is not designed to create views, although you can create them in Interactive SQL and reference them in the Query Editor. Nor was it designed to create UPDATE statements or other non-SELECT SQL statements. It creates a single SELECT statement, so it does not build unions or intersects of SELECT statements. In addition, the Query Editor does not support Transact-SQL syntax.

See also

- "Querying data" [*SQL Anywhere Server - SQL Usage*].
- "SELECT statement" [*SQL Anywhere Server - SQL Reference*].

Using the Expression Editor

The Expression Editor helps you create search conditions, computed columns, and subqueries. To edit existing expressions, highlight the expression before you open the Expression Editor. Otherwise, what you create in the Expression Editor is appended to previously existing expressions when you click **OK**.

Components

- **Expression** This box is where you build your expression.
- **Columns** This box lists the columns that are in your query. To insert a column into your expression, double-click it here, or type it directly into the **Expression** pane.
- **Functions** Functions are predefined expressions used to return information about the database. To insert a function into your expression, choose it from the dropdown list. See "SQL functions" [*SQL Anywhere Server - SQL Reference*].
- **Stored Procedures** This box lists the stored procedures that you can use.
- **Numeric keypad** The numeric keypad is located in the bottom left section of the window. Click a number to insert it into the expression. You can also type numbers from your keyboard.
- **Comparison operators** These arithmetic symbols, such as =, appear in the middle of the bottom section of the window. Click a symbol to insert it into the expression. Only a subset of common operators are included here, but you can type others from your keyboard. See "Comparison operators" [*SQL Anywhere Server - SQL Reference*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Logical operators** Logical operators, such as AND, appear in the right bottom section of the window. Click an operator to insert it into the expression. This group is a subset of commonly used logical operators. You can use any logical operator by typing it from your keyboard. See "Logical operators" [*SQL Anywhere Server - SQL Reference*].

- **Edit a subquery and insert the results into the editor button** This button is located with the logical operators. Click it to access a window that guides you in creating subqueries.

Adding a subquery

To create a subquery, open the Expression Editor and click the **Edit a subquery and insert the results into the editor** button (located next to the **NOT** button). This opens the **Subquery** window, which looks identical to the main Query Editor except that the word **subquery** is in the title bar. You create a subquery just as you create a main query.

Editing a subquery

To edit a subquery in the Query Editor, highlight the subquery in the **Criteria** box, and click the **Expression Editor** button. Highlight the code in the **Expression** pane, and click the **Subquery** button.

See also

- For more information about the Query Editor, see "Using the Query Editor" on page 720.
- For more information about search conditions, see "Search conditions" [*SQL Anywhere Server - SQL Reference*].
- For more information about subqueries, see "Subqueries in search conditions" [*SQL Anywhere Server - SQL Reference*].

Using the Plan Viewer to view graphical plans

The Plan Viewer is a graphical tool for viewing graphical plans for SQL Anywhere databases and text plans for UltraLite databases. See "View an UltraLite execution plan" [*UltraLite - Database Management and Reference*].

To create a graphical plan with detailed and node statistics

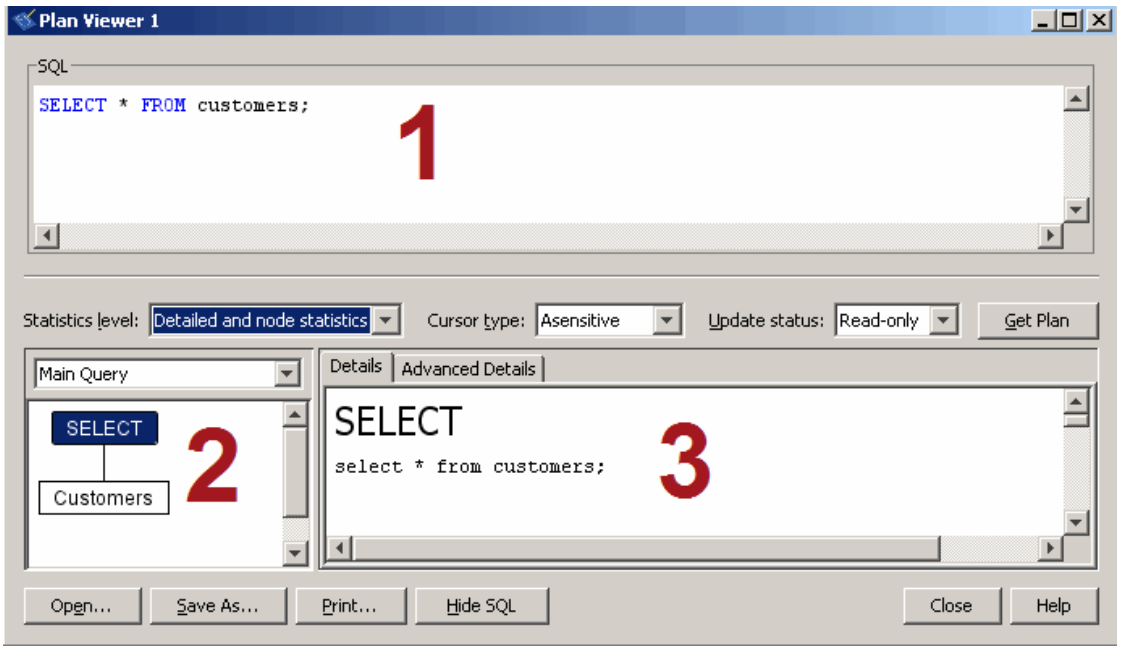
1. Open Interactive SQL and connect to the database.
2. Choose **Tools** » **Plan Viewer** or press Shift+F5.

   The Plan Viewer appears in a separate window.
3. Type a statement in the **SQL** pane.
4. In the **Statistics level** list, click **Detailed and node statistics**.
5. Click **Get Plan** to generate a plan for the specified query.
6. Click **Save As**.

7. Specify where you want to save the plan and type a file name. Click **Save**.

The Plan Viewer window is divided into panes:

- **SQL pane** This pane provides a place for you to type SQL statements that you want to generate plans for.
- **Results pane** This pane shows the graphical plan. This pane is only for SQL Anywhere databases.
- **Details pane** This pane provides text details about the plan for SQL Anywhere databases. For UltraLite databases, this pane shows the text plan.



To open a graphical plan

1. Choose **Tools** » **Plan Viewer**.
2. Click **Open**.
3. Select a plan file (*.saplan*), and then click **Open**.

See also

- "Reading graphical plans" [*SQL Anywhere Server - SQL Usage*]

Configuring the graphical plan

After executing the graphical plan you can customize the appearance of items in the plan.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To change the appearance of the graphical plan

1. Right-click the plan in the lower left pane of the Plan Viewer and choose **Customize**.
2. Change the settings.
3. Click **OK** when finished.
4. Click **Get Plan** generate the graphical plan with your changes.

See also

- "Customizing the appearance of graphical plans" [*SQL Anywhere Server - SQL Usage*]

Viewing graphical images using the Interactive SQL Spatial Viewer

See "View spatial data as images" [*SQL Anywhere Server - Spatial Data Support*].

Viewing images in Interactive SQL

You can use Interactive SQL to view BLOB data, as well as images that are less than 5000000 pixels. In addition you can view SVGs and save them to other formats, such as JPEG, PNG, and TIFF.

Supported image formats include: BMP (Windows 95 version), GIF, JPEG, PNG, PNM, SVG, TIFF, and WBMP (wireless bitmap).

To view images and SVGs (Interactive SQL)

1. Connect to the database. For example, connect to the SQL Anywhere sample database.
2. Execute a query that returns a result set that contains an image.

   For example, run the following to return JPEG images:

   ```
   SELECT * FROM products;
   ```

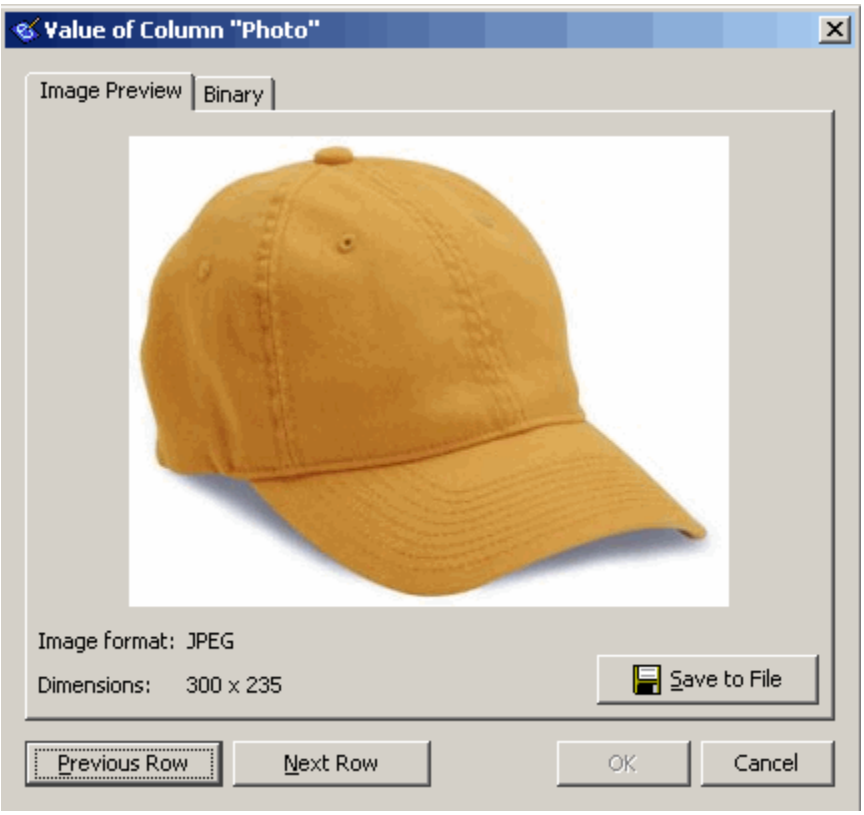
   Run the following to return an SVG image:

   ```
   SELECT '<?xml version="1.0" standalone="no"?>
   <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

   <svg width="100%" height="100%" version="1.1"
   xmlns="http://www.w3.org/2000/svg">

   <rect width="99" height="99"
   style="fill:rgb(0,0,255);stroke-width:1;
   stroke:rgb(0,0,0)"/>

   </svg>
   ' -- Don't forget this closing quote
   ```

3. In the result set, click in a cell that contains **(IMAGE)**, click **...**, and then choose **View in Window**.

The **Value of Column *column-name*** window appears.

For example, the following window appears when you click the JPEG image associated with the cotton cap:



When viewing SVGs, use the following shortcuts:

Shortcut(s)	Description
CTRL+Left Mouse Button Drag	Drags a boundary so that you can adjust the point of view.
CTRL+Right Mouse Button Drag	Rotates the image around its center.
Shift+Left Mouse Button Drag	Pans the image to a new location.
Shift + Right Mouse Button Drag	Zooms the image in and out.

4. Click **Cancel** to close the window.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Viewing HTML and XML data in Interactive SQL

You can view HTML and XML data in Interactive SQL.

To view HTML and XML data (Interactive SQL)

1. Connect to the database.
2. Execute a query that returns a result set that contains HTML and XML data.

   For example to view a result set that contains HTML, run the following:

```
select '
<html>
 <head>
  <meta http-equiv=Content-Type content="text/html; charset=windows-1252">
   <title>Tee Shirt</title>
 </head>
 <body lang=EN-US>
  <p>
   <span  style=font-size:10.0pt;font-family:Arial>
   We have improved the design of this perennial favorite. A sleek and
technical shirt built for the trail, track, or sidewalk. UPF rating of
50+.
   </span>
  </p>
 </body>
</html>
'
```

   For example to view a result set that contains XML, run the following:

```
SELECT '<?xml version="1.0" encoding="UTF-8"?>

<parent>
    <children>
        <child name="Tim" />
        <child name="Katie" />
    </children>
</parent>

'
```

3. In the result set, select a cell that contains HTML content, click **...**, and then choose **View in Window**.

   The **Value of Column *column-name*** window appears.

4. Click **Cancel** to close the window.

Printing SQL statements, execution plans, and result sets

You can print the contents of the **SQL Statements** pane or query results by:

- Pressing Ctrl+P

- Choosing **File** » **Print**.

You can print a plan in the Plan Viewer by:

- Clicking **Print**.
- Right-clicking the plan, and choosing **Print**.

You can also add a header or footer and configure other formatting options in the Interactive SQL **Options** window.

To add a header

1. Open the Interactive SQL **Options** window.

   Choose **Tools** » **Options**.

2. On the **Editor** page click the **Print** tab.

3. In the **Header** field, specify the text that you want to appear in the header. You can also click the right arrow and choose items to include in the header.

Opening multiple windows

You can open multiple Interactive SQL windows. Each window corresponds to a separate database connection. You can connect simultaneously to two (or more) different databases on different database servers, or you can open concurrent connections to a single database.

To open a new Interactive SQL window

1. From the **Window** menu, choose **New Window**.

   Tip
   If the SQLCONNECT environment variable is set, or if you are already connected to a SQL Anywhere database, the database server attempts to use this information to connect to a database before it prompts you for information. Likewise, if the ULSQLCONNECT environment variable is set, or if you are already connected to an UltraLite database, the database server attempts to use this information to connect to a database before it prompts you for information in. If these attempts fail, or if you are not already connected to a database, the **Connect** window appears.

2. In the **Connect** window, enter the connection information for your database, and click **Connect**.

   The connection information (including the database name, your user ID, and the database server name) appears in the Interactive SQL title bar.

You can also connect to or disconnect from a database with the **Connect** and **Disconnect** items in the **SQL** menu, or by executing a CONNECT or DISCONNECT statement.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "SQLCONNECT environment variable" on page 389
- "Storing UltraLite parameters with the ULSQLCONNECT environment variable" [*UltraLite - Database Management and Reference*]

Using source control integration

Interactive SQL integrates with third-party source control systems, allowing you to perform many common source control operations on files from within Interactive SQL. On Windows, Interactive SQL integrates with most source control products that support the Microsoft Common Source Code Control API (SCC), including Microsoft Visual SourceSafe. To use source control products that do not support the SCC API on Windows and other operating systems, specify a command line to run for each of the source control actions. Output from those commands appears in a log window.

Interactive SQL supports the following tasks (as long as the task is supported in the source control product):

- Open a source control project
- Get
- Check in
- Check out
- Undo check out
- Compare versions
- Show file history
- Show file properties
- Run the source control manager

If the underlying source control program does not support an action, its corresponding menu item is disabled. For example, Microsoft Visual SourceSafe supports all of these actions, but using a custom (command line) source control system does not support opening a source control project, or running a source control manager.

You should be familiar with the operations of your source control program before attempting to use it from Interactive SQL.

Configuring Interactive SQL to use source control

You must configure Interactive SQL to use source control before you can perform source control actions on files, such as checking files in and out, comparing different versions of a file, and viewing the history for a file.

If you are running Interactive SQL on a Windows computer that has a source control product that supports the Microsoft SCC API, you can use that product or use a custom (command line oriented) system.

Configuring SCC source control systems

To configure Interactive SQL source control on Windows with SCC

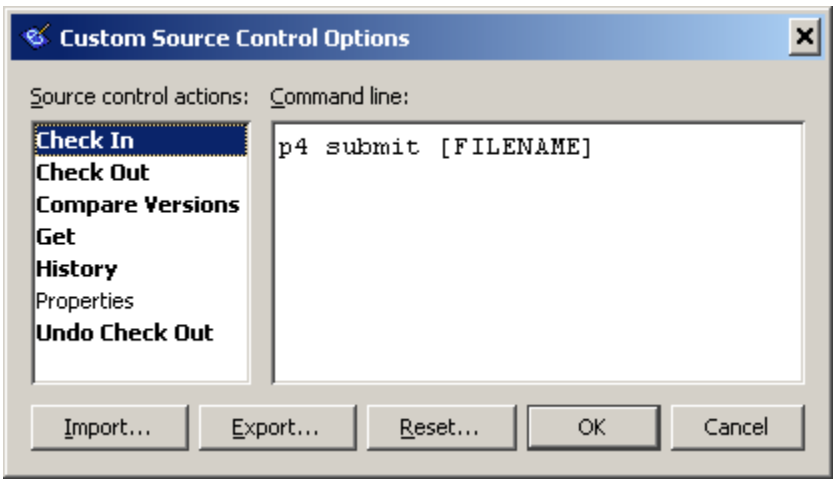
1. Click **Tools** » **Options**.
2. In the left pane, click **Source Control**.
3. Click **Enable Source Control Integration**.
4. Click **OK**.

Configuring other source control systems

To configure Interactive SQL source control systems with a command line interface

1. Click **Tools** » **Options**.
2. In the left pane, click **Source Control**.
3. Click **Enable Source Control Integration**.
4. Click **Configure**.
5. In the **Custom Source Control Options** window, click **Reset**.
6. Select your source control system from the list, and then click **OK**.
7. Edit the commands in the list as necessary by selecting an action from the **Source Control Actions** list, and then typing the corresponding command in the **Command Line** pane.

   When you are defining commands for your system in the **Source Control Actions** list, use the placeholder [FILENAME] to represent the name of the file that is used when you run the command. For example, the command to submit a file in Perforce is `p4 submit [FILENAME]`. Actions that appear bold in this list have commands defined for them, while actions in plain font do not have a command defined.



 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

If you do not specify a command line for an action, the item in the **File** » **Source Control** menu is disabled.

Tip
You can export your source control command lines to an external file by clicking **Export** in the **Custom Source Control Options** window (accessed by choosing **Tools** » **Options**, and then clicking **Configure** on the **Source Control** pane). You can later read these commands back in by clicking **Import** in this window. This may be useful if you need to configure Interactive SQL source control command lines on several computers.

8. Click **OK**, and then click **OK** again.

Opening source control projects from Interactive SQL

Some source control products require you to open a source control project before you can perform any other source control actions. The exact definition of what a project is depends on the source control system you are using. Typically, it is a set of files that are under source control, along with a location on your local file system where working copies of the files are placed. You usually have to provide some credentials, such as a user ID and password, to the source control system to open a project.

If your source control system supports opening a source control project, the **File** » **Source Control** » **Open Source Control Project** menu item is enabled. Choosing this option from the **File** menu opens a source control-specific window for opening a project. Once you open a project, you do not have to open it again, even in subsequent Interactive SQL sessions. The project is opened automatically for you.

Checking files out from Interactive SQL

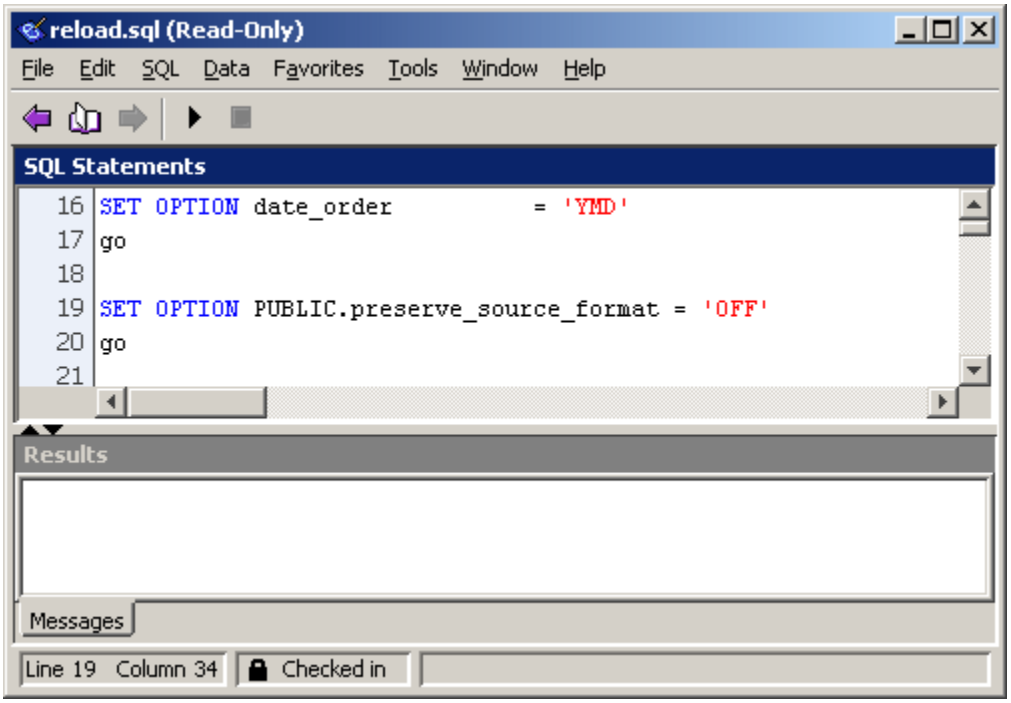
Once you open a file in Interactive SQL, there are two ways you can check the file out: modifying its contents in the **SQL Statements** pane, or using the command on the **File** menu.

When you configure the source control options for Interactive SQL, if you select **Automatically Check Out Files When Editor Contents Are Modified**, Interactive SQL attempts to check out a file when you modify its contents in the **SQL Statements** pane.

To check out a file using the Interactive SQL File menu

1. Choose **File** » **Open**, and then browse to the file you want to open.

   The file status appears on the status bar at the bottom of the Interactive SQL window. The status is one of **Checked In**, **Checked Out**, or **Not Controlled**. Files that are checked in are assumed to be read-only, and **Read-Only** appears in the Interactive SQL title bar. The file in the following example is checked in:



2. Check out the file by choosing **File** » **Source Control** » **Check Out**.

   Depending on which source control product you are using, you may be prompted for a comment or other options as part of the check out procedure.

Caution
If you are using a SCC-compliant source control system, the status is always accurate. However, if you use the custom source control system, the status is based on whether the file is read-only or not. A read-only file is assumed to be checked in, but no assumptions are made about editable files because they could be either checked out or not controlled.

Checking in files from Interactive SQL

When you are finished making edits to your file, you can check it back in from Interactive SQL.

To check in a file from Interactive SQL

1. Choose **File** » **Source Control** » **Check In**.
2. Enter check in comments if you are prompted.

Additional source control actions

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

In addition to opening source control projects, and checking files in and out, Interactive SQL supports several other source control actions. The availability of these actions depends on the source control system you are using. These actions are accessed from the **File** » **Source Control** menu in Interactive SQL.

- **Get** This action gets the latest copy of the file you currently have open in the **SQL Statements** pane.
- **Undo Check Out** If you have checked out a file, and you want to discard your changes, choose **File** » **Source Control** » **Undo Checkout**. This discards your working copy of the file, and then downloads the copy of the file that is in the source control archive.
- **Compare Versions** This action compares the working copy of the file you have opened against the version in the source control archive.
- **History** This action displays a list of source control actions (typically check-ins) that have been made to the file you have open.
- **Properties** This action displays a list of source control properties that are associated with the file you have opened.
- **Run Source Control Manager** This action launches the management program for your source control system. For example, if you are using Microsoft Visual SourceSafe, this launches Microsoft Visual SourceSafe Explorer.

Interactive SQL SQL statements

The following list provides links into the SQL Anywhere documentation for the SQL statements that are available to users of Interactive SQL.

"CLEAR statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"CONFIGURE statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"CONNECT statement [ESQL] [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"DESCRIBE statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"DISCONNECT statement [ESQL] [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"EXIT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"HELP statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"OUTPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"PARAMETERS statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"READ statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"SET CONNECTION statement [Interactive SQL] [ESQL]" [*SQL Anywhere Server - SQL Reference*]

"SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"START SERVER statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"START LOGGING statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"STOP LOGGING statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

"SYSTEM statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

See also

- "SQL statements" [*SQL Anywhere Server - SQL Reference*]
- "UltraLite SQL statements" [*UltraLite - Database Management and Reference*]

Interactive SQL keyboard shortcuts

Interactive SQL provides the following keyboard shortcuts:

Key(s)	Description
Alt+F4	Shuts down Interactive SQL.
Alt+Left Arrow On Mac OS X, use Control Commmand+Left Arrow.	Displays the previous SQL statement in the history list.
Alt+Right Arrow On Mac OS X, use Control Commmand+Right Arrow.	Displays the next SQL statement in the history list.
Ctrl+Backspace	Deletes the word to the left of the cursor.
Ctrl+Break	Interrupts the SQL statement that is being executed.
Ctrl+A	Selects all text in the active pane. In the **Results** pane, if the results are not the entire result set, you are prompted to fetch the remaining rows. Otherwise the currently fetched results are selected. See "Copying columns, rows, and cells from an Interactive SQL result set" on page 719.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Key(s)	Description
Ctrl+C	In the **Results** pane, copies the selected row(s) and column headings to the clipboard. In the **SQL Statements** pane, copies the selected text to the clipboard.
Ctrl+Shift+C	Executes a Commit statement. See "Executing COMMIT and ROLLBACK statements in Interactive SQL" on page 708.
Ctrl+Del	Deletes the word to the right of the cursor.
Ctrl+End	Moves the cursor to the bottom of the current pane.
Ctrl+F	Opens the **Find/Replace** window.
CTRL+SHIFT+F10	Displays a popup menu for a cell that has focus in a result set in the **Results** tab. This keyboard shortcut is an alternative to clicking **...** within a table cell.
Ctrl+G	Moves the cursor to the specified line in the **SQL Statements** pane.
Ctrl+H On Mac OS X, use Control Commmand+H.	Displays the history of your executed SQL statement(s).
Ctrl+Home	Moves the cursor to the top of the current pane.
Ctrl+L	Deletes the current line from the **SQL Statements** pane and puts the line onto the clipboard.
Ctrl+Shift+L	Deletes the current line.
Ctrl+N	Clears the contents of the Interactive SQL window and closes the current file (if any).
Ctrl+O	Opens a file.
Ctrl+P	Prints the contents of the **SQL Statements** pane; from the **Tools** menu, choose **Options**, then choose **Editor**, and then click the **Print** tab.
Ctrl+Q	Opens the Query Editor. The Query Editor helps you build SQL queries. When you have finished building your query, click **OK** to export it back into the **SQL Statements** pane.

Key(s)	Description
Ctrl+Shift+R	Executes a Rollback statement. See "Executing COMMIT and ROLLBACK statements in Interactive SQL" on page 708.
Ctrl+S	Saves the contents of the **SQL Statements** pane to the specified file.
Ctrl+U	Changes the selection to lowercase characters.
Ctrl+Shift+U	Changes the selection to uppercase characters.
Ctrl+V	Pastes the selected text.
Ctrl+X	Cuts the selected text.
Ctrl+Y	Repeats the last operation.
Ctrl+Z	Undoes the last operation.
Ctrl+]	Moves the cursor to the matching brace. Use this shortcut to match parentheses, braces, brackets, and angle brackets.
Ctrl+Shift+]	Extends the selection to the matching brace. Use this shortcut to match parentheses, braces, brackets, and angle brackets.
Ctrl+Minus Sign (-)	Adds and removes the double-hyphen (--) SQL comment indicator. To turn existing text into comments, select the text in the **SQL Statements** pane and press Ctrl+minus sign. The SQL comment indicator is added to the beginning of each line in the selection. If no text is selected, the comment indicator is added to the beginning of the current line. To remove a comment indicator, select the text and press Ctrl+minus sign. See "Comments" [*SQL Anywhere Server - SQL Reference*].
Ctrl+Forward Slash (/)	Adds and removes the double-slash (//) SQL comment indicator. To turn existing text into comments, select the text in the **SQL Statements** pane and press Ctrl+forward slash. The SQL comment indicator is added to the beginning of each line in the selection. If no text is selected, the comment indicator is added to the beginning of the current line. To remove a comment indicator, select the text and press Ctrl+forward slash. See "Comments" [*SQL Anywhere Server - SQL Reference*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Key(s)	Description
Ctrl+Up Arrow	Selects the SQL statement preceding the statement that contains the cursor in the **SQL Statements** pane.
Ctrl+Down Arrow	Selects the SQL statement following the statement that contains the cursor in the **SQL Statements** pane.
Ctrl+Period (.)	Selects the entire SQL statement containing the cursor in the **SQL Statements** pane.
Ctrl+Comma (,)	Selects the line containing the cursor in the **SQL Statements** pane.
Ctrl+Shift+Period (.)	Increases the line indentation of selected text in the **SQL Statements** pane. If no text is selected, the indentation is applied to the current line. You can change the number of spaces that are indented; from the **Tools** menu, choose **Options**, then choose **Editor**, and then click the **Tabs** tab.
Ctrl+Shift+Comma (,)	Decreases line indentation of selected text in the **SQL Statements** pane. If no text is selected, the indentation is applied to the current line. You can change the number of spaces that are indented; from the **Tools** menu, choose **Options**, then choose **Editor**, and then click the **Tabs** tab.
Esc	By default, pressing the Esc key has no affect. However, you can set the Esc key to clear the **SQL Statements** pane and close any opened result sets. Choose **Tools** » **Options** » **Compatibility** and select **ESCAPE key Clears SQL Statements And Closes Result Sets**.
F1	Opens online help.
F2	Edits the selected value in the result set. You can tab from column to column within the row.
F3	Finds the next occurrence of the specified text.
Shift+F3	Finds the previous occurrence of the selected text.
F5	Executes all text in the **SQL Statements** pane. You can also perform this operation by clicking **Execute All SQL Statements** on the toolbar or choosing **SQL** » **Execute**.

Key(s)	Description
Shift+F5	Opens the Plan Viewer for the specified statement in the **SQL Statements** pane. The specified statement is not executed; to execute the statement in the Plan Viewer, click **Get Plan**.
F6	Changes the focus from the **SQL Statements** pane to the **Results** pane and vice versa. When the **Favorites** pane is opened, the focus changes between the **SQL Statements**, **Results**, and **Favorites** panes.
F7	Displays the **Lookup Table Name** window. In this window, you can find and select a table and then press Enter to insert the table name into the **SQL Statements** pane at the cursor position. Or, with a table selected in the list, press F7 again to display the columns in that table. You can then select a column and press Enter to insert the column name into the **SQL Statements** pane at the cursor position.
F8	Displays the **Lookup Procedure Name** window. In this window, you can find and select a procedure and then press Enter to insert the procedure name into the **SQL Statements** pane at the cursor position.
F9	Executes the text that is selected in the **SQL Statements** pane. If no text is selected, all the statements are executed. You can also perform this operation by clicking **Execute Selected Statements** on the toolbar or choosing **SQL** » **Execute**.
Shift+F9	Executes the selected SQL statement, and then selects the next statement. This shortcut allows you to step through a series of SQL statements. See "Executing SQL statements from Interactive SQL" on page 705.
Shift+F10	Displays the shortcut menu for the area that has focus. This keyboard shortcut is an alternative to right-clicking an area.
F11	Opens the **Connect** window if Interactive SQL is not connected to a database.
F12	Disconnects Interactive SQL from the current database.
Home	Moves the cursor to the start of the current line or to the first word on the current line.
Shift+Home	Extends the selection to the start of the text on the current line.
Page Down	Moves a page down in the current pane.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Key(s)	Description
Page Up	Moves a page up in the current pane.

Interactive SQL options

Use the SET OPTION statement to change the values of the following Interactive SQL options. See "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*].

Option	Values	Default
"auto_commit option [Interactive SQL]" on page 740	On, Off	Off
"auto_refetch option [Interactive SQL]" on page 741	On, Off	On
"bell option [Interactive SQL]" on page 741	On, Off	On
"command_delimiter option [Interactive SQL]" on page 742	String	' ; '
"commit_on_exit option [Interactive SQL]" on page 743	On, Off	On
"default_isql_encoding option [Interactive SQL]" on page 743	String	Empty string
"echo option [Interactive SQL]" on page 744	On, Off	On
"input_format option [Interactive SQL]" on page 745	TEXT, FIXED	TEXT
"isql_allow_read_client_file option [Interactive SQL]" on page 745	On, Off, Prompt	Prompt
"isql_allow_write_client_file option [Interactive SQL]" on page 746	On, Off, Prompt	Prompt
"isql_command_timing option [Interactive SQL]" on page 747	On, Off	On
"isql_escape_character option [Interactive SQL]" on page 747	Character	' \ '

Option	Values	Default
“isql_field_separator option [Interactive SQL]” on page 748	String	' , '
“isql_maximum_displayed_rows option [Interactive SQL]” on page 749	All or a non-negative integer	500
“isql_print_result_set option [Interactive SQL]” on page 750	Last, All, None	Last
“isql_quote option [Interactive SQL]” on page 750	String	'
“isql_show_multiple_result_sets [Interactive SQL]” on page 751	On, Off	Off
“nulls option [Interactive SQL]” on page 752	String	'(NULL)'
“on_error option [Interactive SQL]” on page 752	Stop, Continue, Prompt, Exit, Notify_Continue, Notify_Stop, Notify_Exit	Prompt
“output_format option [Interactive SQL]” on page 753	TEXT, FIXED, HTML, SQL, XML	TEXT
“output_length option [Interactive SQL]” on page 754	Integer	0
“output_nulls option [Interactive SQL]” on page 754	String	Empty string
“truncation_length option [Interactive SQL]” on page 754	Integer	256

auto_commit option [Interactive SQL]

Controls whether a COMMIT is performed after each statement.

Allowed values

On, Off

Default

Off

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

If auto_commit is On, a database COMMIT is performed after each successful statement.

By default, a COMMIT or ROLLBACK is performed only when the user issues a COMMIT or ROLLBACK statement or a SQL statement that causes an automatic commit (such as the CREATE TABLE statement).

Using a data source in Interactive SQL
By default, ODBC operates in autocommit mode. Even if you have set the auto_commit option to Off in Interactive SQL, the ODBC setting overrides the Interactive SQL setting. You can change ODBC's setting using the SQL_ATTR_AUTOCOMMIT connection attribute. ODBC autocommit is independent of the chained option.

See also

- "commit_on_exit option [Interactive SQL]" on page 743
- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "Executing COMMIT and ROLLBACK statements in Interactive SQL" on page 708

auto_refetch option [Interactive SQL]

Controls whether query results are fetched again after deletes, updates, or inserts.

Allowed values

On, Off

Default

On

Remarks

If auto_refetch is On, the current query results that appear on the **Results** tab in the Interactive SQL **Results** pane are refetched from the database after any INSERT, UPDATE, or DELETE statement. Depending on how complicated the query is, this may take some time. For this reason, it can be turned off.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

bell option [Interactive SQL]

Controls whether the bell sounds when an error occurs.

Allowed values

On, Off

Default

On

Remarks

Set this option according to your preference.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

command_delimiter option [Interactive SQL]

Sets the string that indicates the end of a statement in Interactive SQL.

Allowed values

String

Default

Semicolon (;)

Remarks

In general, there is no need to change the command delimiter. You should leave it as a semicolon.

An alternative to using a semicolon or another string as a statement delimiter is to enter the separator **go** on a line by itself, at the beginning of the line. See "Introduction to batches" [*SQL Anywhere Server - SQL Usage*].

Specifying **go** on its own line at the beginning of the line is always recognized as a command delimiter, even if you set the command_delimiter option to a different value.

The command_delimiter value can be any string of characters with the following restrictions:

- If the delimiter contains any one of & (ampersand), * (asterisk), @ (at sign), : (colon), . (period), = (equals), ((left parentheses),) (right parentheses), or | (vertical bar), the delimiter must not contain any other character. For example, * is a valid delimiter, but ** is not.
- You should not use an existing keyword as a command separator. See "Keywords" [*SQL Anywhere Server - SQL Reference*].
- The command delimiter can be any sequence of characters (including numbers, letters, and punctuation), but it cannot contain embedded blanks. As well, it can contain a semicolon, but only as the first character.

  If the command delimiter is set to a string beginning with a character that is valid in identifiers, the command delimiter must be preceded by a space. The command delimiter is case sensitive. You must enclose the new command delimiter in single quotation marks. When the command delimiter is a semicolon (the default), no space is required before the semicolon.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Interactive SQL utility (dbisql)" on page 812
- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

Examples

The following example sets the command delimiter to a tilde:

```
SET OPTION command_delimiter='~';
MESSAGE 'hello'~
```

You can also use the Interactive SQL -d option to set the command delimiter without including a SET OPTION command_delimiter statement in a *.sql* file. For example, if you have a script file named *test.sql* that uses tildes (~) as the command delimiter, you could run:

```
dbisql -d "~" test.sql
```

commit_on_exit option [Interactive SQL]

Controls the behavior when Interactive SQL disconnects or shuts down.

Allowed values

On, Off

Default

On

Remarks

Controls whether a COMMIT or ROLLBACK is done when you leave Interactive SQL. When commit_on_exit is set to On, a COMMIT is done.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "Executing COMMIT and ROLLBACK statements in Interactive SQL" on page 708

default_isql_encoding option [Interactive SQL]

Specifies the encoding that should be used by READ, INPUT, and OUTPUT statements.

Allowed values

Identifier or string

Default

Use system encoding (empty string)

Scope

Can be set as a temporary option only, for the duration of the current connection.

Remarks

This option is used to specify the encoding to use when reading or writing files. It cannot be set permanently. The default encoding is the code page for the platform you are running on. On English Windows computers, the default code page is 1252.

When running Interactive SQL, the encoding that is used by the INPUT, OUTPUT, or READ statement is determined in the following order:

- The encoding specified by the ENCODING clause of the INPUT, OUTPUT, or READ statement (if this clause is specified).
- The encoding specified with the default_isql_encoding option (if this option is set).
- The default encoding for the platform you are running on. On English Windows computers, the default encoding is 1252.

If the input file was created using the OUTPUT statement and an encoding was specified, then the same ENCODING clause should be specified on the INPUT statement. See “OUTPUT statement [Interactive SQL]” [*SQL Anywhere Server - SQL Reference*].

For more information about code pages and character sets, see “International language and character set tasks” on page 430.

See also

- “READ statement [Interactive SQL]” [*SQL Anywhere Server - SQL Reference*]
- “INPUT statement [Interactive SQL]” [*SQL Anywhere Server - SQL Reference*]
- “OUTPUT statement [Interactive SQL]” [*SQL Anywhere Server - SQL Reference*]
- “Understanding character sets” on page 405
- “SET OPTION statement [Interactive SQL]” [*SQL Anywhere Server - SQL Reference*]

Example

Set the encoding to UTF-16 (for reading Unicode files):

```
SET TEMPORARY OPTION default_isql_encoding = 'UTF-16';
```

Set the encoding to cp437 I (Windows English OEM):

```
SET TEMPORARY OPTION default_isql_encoding = 'cp437';
```

echo option [Interactive SQL]

Controls whether statements are echoed to the log file before they are executed.

Allowed values

On, Off

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Default

On

Remarks

This option is most useful when you use the READ statement to execute an Interactive SQL command file or when you run a command file in Interactive SQL by choosing **File** » **Run Script**. Logging must be turned on for this option to take effect. See "START LOGGING statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*].

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

input_format option [Interactive SQL]

Sets the default data format expected by the INPUT statement.

Allowed values

- **TEXT** Input lines are assumed to be text characters, one row per line, with values separated by commas. Alphabetic strings can be enclosed in apostrophes (single quotes) or quotation marks (double quotes).

  The default delimiter is a comma (,). Strings containing commas must be enclosed in either single or double quotes. If single or double quotes are used, double the quote character to use it within the string. Optionally, you can use the DELIMITED BY clause to specify a different delimiter string.

  Three other special sequences are also recognized. The two characters **\n** represent a newline character, \\ represents a single backslash character, and the sequence **\x***DD*, where *DD* is the hexadecimal representation of a character, represents the character with hexadecimal code DD.

- **FIXED** Input lines are in fixed length format.

Default

TEXT

See also

- "INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

isql_allow_read_client_file option [Interactive SQL]

Controls whether client file reads are permitted for the connection.

Allowed values

On, Off, Prompt

Default

Prompt

Remarks

This option controls whether the database server can read files on the client computer. On means reading is allowed. Off means reading is not allowed. Prompt means prompt the user for the action to take.

This option is stored on a per-connection basis and persists only for the duration of the connection. You can set this option using the SET TEMPORARY OPTION statement. If you omit the TEMPORARY keyword, Interactive SQL reports an error.

This option allows a data file to be read without user intervention when a LOAD TABLE is executed from a stored procedure or trigger.

READCLIENTFILE authority is required to read a file on a client computer.

See also

- "Accessing data on client computers" [*SQL Anywhere Server - SQL Usage*]
- "READCLIENTFILE authority" on page 446
- "READ_CLIENT_FILE function [String]" [*SQL Anywhere Server - SQL Reference*]
- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "allow_read_client_file option" on page 506
- "allow_write_client_file option" on page 508
- "isql_allow_write_client_file option [Interactive SQL]" on page 746
- "Client-side data security" [*SQL Anywhere Server - SQL Usage*]

isql_allow_write_client_file option [Interactive SQL]

Controls whether client file writes are permitted for the connection.

Allowed values

On, Off, Prompt

Default

Prompt

Remarks

This option controls whether the database server can write to files on the client computer. On means writing is allowed. Off means writing is not allowed. Prompt means prompt the user for the action to take.

This option is stored on a per-connection basis and persists only for the duration of the connection. You can set this option using the SET TEMPORARY OPTION statement. If you omit the TEMPORARY keyword, Interactive SQL reports an error.

WRITECLIENTFILE authority is required to write a file on a client computer.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Accessing data on client computers" [*SQL Anywhere Server - SQL Usage*]
- "WRITECLIENTFILE authority" on page 447
- "WRITE_CLIENT_FILE function [String]" [*SQL Anywhere Server - SQL Reference*]
- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "UNLOAD statement" [*SQL Anywhere Server - SQL Reference*]
- "allow_write_client_file option" on page 508
- "allow_read_client_file option" on page 506
- "isql_allow_read_client_file option [Interactive SQL]" on page 745
- "Client-side data security" [*SQL Anywhere Server - SQL Usage*]

isql_command_timing option [Interactive SQL]

Controls whether SQL statements are timed or not.

Allowed values

On, Off

Default

On

Remarks

This boolean option controls whether SQL statements are timed or not. If you set the option to On, the time of execution appears in the **Messages** pane after you execute a statement. If you set the option to Off, the time does not appear.

Execution times are included in the log file when logging and execution time reporting are both enabled. See "START LOGGING statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*].

The isql_command_timing option appears as the **Measure Execution Times For SQL Statements** option on the **Messages** tab. Choose **Tools** » **Options** » **Messages**, and then select or clear the **Measure Execution Times For SQL Statements** option.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

isql_escape_character option [Interactive SQL]

Controls the escape character used in place of unprintable characters in data exported to text files.

Allowed values

Any single character

Default

A backslash (\ )

Remarks

When Interactive SQL exports strings that contain unprintable characters (such as a carriage return), it converts each unprintable character into a hexadecimal format and precedes it with an escape character. The character you specify for this setting is used in the output if your OUTPUT statement does not contain an ESCAPE CHARACTER clause. This setting is used only if you are exporting to a text file.

See also

- "isql_quote option [Interactive SQL]" on page 750
- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

Example

Create a table that contains one string value with an embedded carriage return (denoted by the "\n" in the INSERT statement). Then export the data to *c:\escape.txt* with a # sign as the escape character.

```
CREATE TABLE escape_test( text varchar(10 ) );
INSERT INTO escape_test VALUES( 'one\ntwo' );
SET OPTION isql_escape_character='#';
SELECT * FROM escape_test;
OUTPUT TO c:\escape.txt FORMAT TEXT;
```

This code places the following data in *escape.txt*:

```
'one#x0atwo'
```

The pound sign (#) is the escape character and **x0a** is the hexadecimal equivalent of the **\n** character.

The start and end characters (in this case, single quotation marks) depend on the isql_quote setting.

isql_field_separator option [Interactive SQL]

Controls the default string used for separating values in data exported to text files.

Allowed values

String

Default

A comma (,)

Remarks

Controls the default string used for separating (or delimiting) values in data exported to text files. If an OUTPUT statement does not contain a DELIMITED BY clause, the value of this setting is used.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "isql_quote option [Interactive SQL]" on page 750
- "isql_escape_character option [Interactive SQL]" on page 747
- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

Example

- The first example sets the field separator to a colon in the data exported to *c:\Employees.txt*.

  ```
  SET OPTION isql_field_separator=':';
  SELECT Surname, GivenName FROM Employees WHERE EmployeeID < 150;
  OUTPUT TO c:\Employees.txt FORMAT TEXT;
  ```

  This code places the following data in *Employees.txt*:

  ```
  'Whitney': 'Fran'
  'Cobb':'Matthew'
  'Chin':'Philip'
  'Jordan':'Julie'
  ```

  The start and end characters (in this case, single quotation marks) depend on the isql_quote setting.

- The next example sets the field separator to a tab in the data exported to *c:\Employees.txt*.

  ```
  SET OPTION isql_field_separator='\t';
  SELECT Surname, GivenName FROM Employees WHERE EmployeeID < 150;
  OUTPUT TO c:\Employees.txt FORMAT TEXT;
  ```

  This code places the following data in *Employees.txt*:

  ```
  'Whitney' 'Fran'
  'Cobb' 'Matthew'
  'Chin' 'Philip'
  'Jordan' 'Julie'
  ```

  The start and end characters (in this case, single quotation marks) depend on the isql_quote setting. The escape character (in this case the backslash) depends on the isql_escape_character setting.

isql_maximum_displayed_rows option [Interactive SQL]

Specifies the maximum number of rows that can appear in the **Results** pane in Interactive SQL.

Allowed values

ALL or a non-negative integer

Default

500

Remarks

This option lets you specify the maximum number of rows that appear in the **Results** pane. You can also set the value for this option in the **Options** window in Interactive SQL.

Caution
Interactive SQL can run out of memory when displaying large result sets. If this problem occurs, Interactive SQL reports the problem but will not display the result set.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

isql_print_result_set option [Interactive SQL]

Specifies which result set(s) are printed when a *.sql* file is run.

Allowed values

- **LAST** Prints the result set from the last statement in the file.
- **ALL** Prints result sets from each statement in the file which returns a result set.
- **NONE** Does not print any result sets.

Default

LAST

Remarks

The isql_print_result_set option takes effect only when you run Interactive SQL as a command line program (for example, when running a *.sql* file).

This option allows you to specify which result set(s) are printed when a *.sql* file is run.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

isql_quote option [Interactive SQL]

Controls the default string that begins and ends all strings in data exported to text files.

Allowed values

String

Default

A single apostrophe (')

Remarks

Controls the default string that begins and ends all strings in data exported to text files. If an OUTPUT statement does not contain a QUOTE clause, this value is used by default.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "isql_field_separator option [Interactive SQL]" on page 748
- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

Example

To change the default string that begins and ends all strings to a double quote character.

```
SET OPTION isql_quote='"';
SELECT Surname, GivenName FROM Employees WHERE EmployeeID < 150;
OUTPUT TO c:\Employees.txt FORMAT TEXT;
```

This code places the following data in *Employees.txt*:

```
"Whitney", "Fran"
"Cobb","Matthew"
"Chin","Philip"
"Jordan","Julie"
```

The separator characters (in this case, commas) depend on the isql_field_separator setting.

isql_show_multiple_result_sets [Interactive SQL]

Specifies whether multiple result sets can appear in the **Results** pane in Interactive SQL.

Allowed values

On, Off

Default

Off

Remarks

Set this option to On if you want Interactive SQL to display multiple result sets in the **Results** pane when you execute a procedure that returns multiple SELECT statements.

Each result set appears on a separate tab in the **Results** pane. By default, Interactive SQL does not display multiple result sets. The setting of this option also applies to Interactive SQL when it is running as a command line program.

The isql_show_multiple_result_sets option also appears as the **Show All Result Sets** option when running Interactive SQL as a windowed application. See "Returning multiple result sets from procedures" [*SQL Anywhere Server - SQL Usage*].

When deploying Interactive SQL, you can prevent users from setting isql_show_multiple_result_sets option. See "showMultipleResultSets (SQLAnywhere/UltraLite)" [*SQL Anywhere Server - Programming*].

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

Examples

The following sets the isql_show_multiple_result_sets to On.

```
SET OPTION isql_show_multiple_result_sets=On
```

nulls option [Interactive SQL]

Specifies how NULL values in the database appear when displaying results in Interactive SQL.

Allowed values

String

Default

(NULL)

Remarks

Set this option according to your preference. Note that this value is not used when saving result sets to a file. The value used when saving to a file is specified by the output_nulls option.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "output_nulls option [Interactive SQL]" on page 754

on_error option [Interactive SQL]

Controls what happens if an error is encountered while executing statements in Interactive SQL.

Allowed values

- **Stop** Interactive SQL stops executing statements.
- **Prompt** Interactive SQL prompts the user to see if they want to continue.
- **Continue** The error is ignored and Interactive SQL continues executing statements.
- **Exit** Interactive SQL shuts down.
- **Notify_Continue** The error is reported, and the user is prompted to press Enter or click **OK** to continue.
- **Notify_Stop** The error is reported and the user is prompted to press Enter or click **OK** to stop executing statements.
- **Notify_Exit** The error is reported and the user is prompted to press Enter or click **OK** to shut down Interactive SQL.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Default

Prompt

Remarks

When you are executing a *.sql* file, the values Stop and Exit are equivalent. If you specify either of these values, Interactive SQL shuts down.

This option is ignored if the Interactive SQL utility (dbisql) onerror option is specified. See "Interactive SQL utility (dbisql)" on page 812.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

output_format option [Interactive SQL]

Sets the default output format for data retrieved by a SELECT statement that is redirected to a file or output using the OUTPUT statement.

Allowed values

- **TEXT** The output is a TEXT format file with one row per line in the file. All values are separated by commas, and strings are enclosed in apostrophes (single quotes). The delimiter and quote strings can be changed using the DELIMITED BY and QUOTE clauses. If All is specified in the QUOTE clause, then all values (not just strings) are quoted.

  Three other special sequences are also used. The two characters **\n** represent a newline character; \\ represents a single backslash character, and the sequence **\x***DD* represents the character with hexadecimal code DD.

- **FIXED** The output is fixed format, with each column having a fixed width. The width for each column can be specified using the COLUMN WIDTH clause. If this clause is omitted, the width for each column is computed from the data type for the column, and is large enough to hold any value of that data type. No column headings are output in this format.

- **HTML** The output is in HTML format.

- **SQL** The output is an Interactive SQL INPUT statement required to recreate the information in the table.

- **XML** The output is an XML file encoded in UTF-8 and containing an embedded DTD. Binary values are encoded in CDATA blocks with the binary data rendered as 2-hex-digit strings.

Default

TEXT

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

output_length option [Interactive SQL]

Controls the length of column values when Interactive SQL exports information to an external file.

Allowed values

Non-negative integer

Default

0 (no truncation)

Remarks

This option controls the maximum length of column values when Interactive SQL exports data to an external file (using output redirection with the OUTPUT statement). This option affects only the TEXT, HTML, and SQL output formats.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

output_nulls option [Interactive SQL]

Controls the way NULL values are exported.

Allowed values

String

Default

Empty string

Remarks

This option controls the way NULL values are written by the OUTPUT statement. Every time a NULL value is found in the result set, the string from this option is returned instead. This option affects only the TEXT, HTML, FIXED, and SQL output formats.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

truncation_length option [Interactive SQL]

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Controls the truncation of wide columns for displays to fit on a screen.

Allowed values

Integer

Default

256

Remarks

The truncation_length option limits the length of displayed column values. The unit is in characters. A value of 0 means that column values are not truncated. The default truncation length is 256.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

Using text completion

Interactive SQL and Sybase Central provide a text completion option that can supply object names for you. You can configure text completion to complete the name of any or all the following object types: tables, views, columns, stored procedures, and system functions.

For SELECT, INSERT, UPDATE, DELETE, and DESCRIBE statements, the list of possible suggestions is relative to where you are typing within the statement. For example, consider the following SQL statement:

```
SELECT EmployeeID FROM Employees as e WHERE e.EmployeeID>=20;
```

If you open the text completion window after SELECT, the list contains column names in the Employees table, and stored procedures and SQL functions.

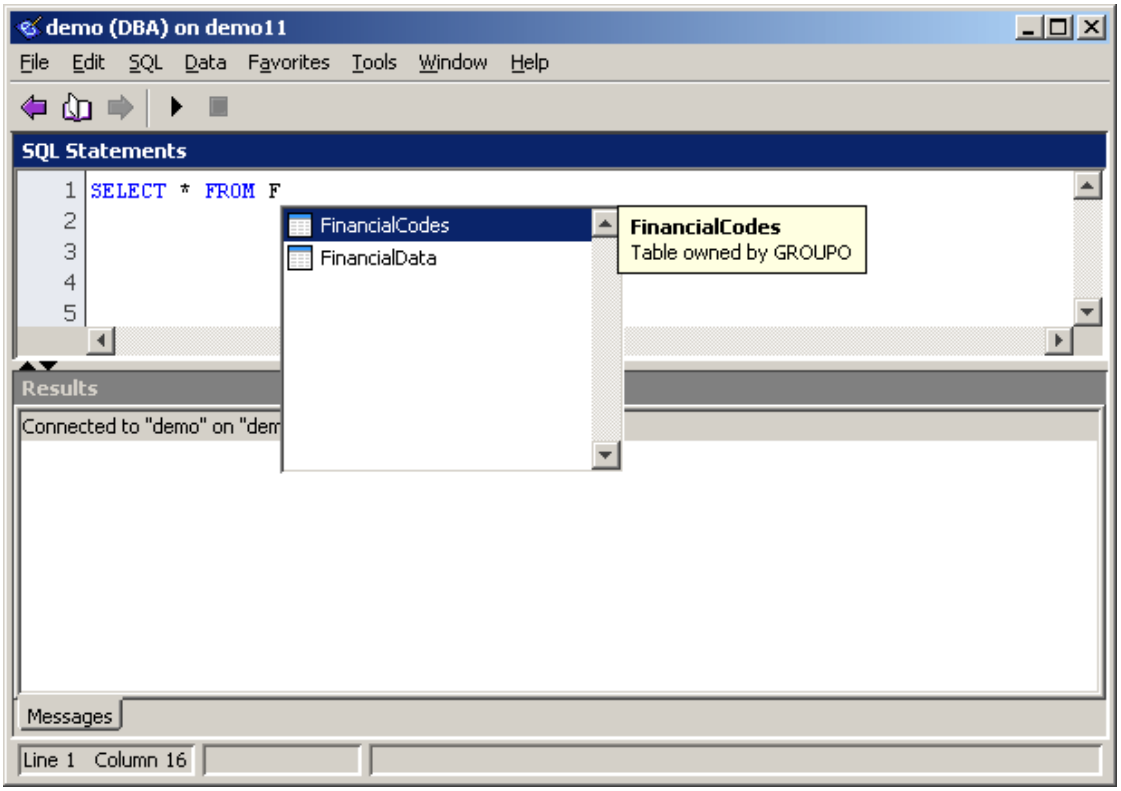
If you open the text completion window after FROM, the list contains only tables and stored procedures.

If you open the text completion window after the e in the WHERE clause, the list contains only columns in the table whose alias is e.

To use text completion

1. In Interactive SQL, type the first letter of a database object name in the **SQL Statements** pane.
2. Press Ctrl+Space or Ctrl+Shift+Space.

   A window appears listing the names of database objects that begin with the letter(s) you typed. In the following example, it shows all database objects that begin with the letter F.



If you do not see the object name you want, press Tab to view a complete list of database objects (based on the filtering options you set—by default, all database objects appear in the list).

3. Select the object name from the list and then press Enter.

   The object name appears in the **SQL Statements** pane.

You can configure the text completion settings from the **Options** window in Interactive SQL or when you are in a text editor window in Sybase Central.

Text completion keyboard shortcuts

The following keyboard shortcuts are available when the text completion list is open.

Key	Description
Ctrl+C	Shows only columns in the text completion list.
Ctrl+F	Shows only SQL functions in the text completion list.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Key	Description
Ctrl+P	Shows only stored procedures and functions in the text completion list.
Ctrl+S	Changes the contents of the list to show or hide system objects.
Ctrl+Shift+Space	Opens the text completion window. You can also use Ctrl+Space to open the text completion window.
Ctrl+T	Shows only tables in the text completion list.
Ctrl+V	Shows only views in the text completion list.
Esc	Closes the text completion window without adding any text.
Tab	Toggles between a list of all database object names and a list of names that match what has been typed so far.
*	For tables, inserts a comma separated list of columns, including data types. For stored procedures, inserts the procedure name, followed by a comma-separated list of parameter names and their data types.
+	For tables, inserts a comma-separated list of columns. For stored procedures, inserts the procedure name, followed by a comma-separated list of parameter names.
"	Completes the name, enclosing it in quotation marks, regardless of the setting of the quoted_identifier option. See "quoted_identifier option" on page 579.

Using the fast launcher option

The fast launcher option reduces the startup time for Sybase Central and Interactive SQL. When fast launching is enabled, the program stays in memory for a configurable length of time after you close it. If you restart the program within this time, it starts quickly. If you do not restart the program within this time, the process terminates and releases its resources to the operating system.

When the fast launcher option is enabled, the program's icon appears in the system tray. You can restart or shut down the program by right-clicking the system tray icon. Using the system tray icon to shut down the program terminates the fast launcher process and releases its resources to the operating system. The fast launcher option is only available on Windows.

Changing language settings

If the fast launcher is still running after Sybase Central or Interactive SQL is shut down, and you change the language settings using the dblang utility the changes do not take effect when Sybase Central is restarted.

To ensure that both utilities are shut down on Windows you can use the Task Manager to end the task. Alternatively, you can wait for the fast launcher inactivity timer to take effect.

Configuring the fast launcher option

The fast launcher option uses a TCP/IP port on your computer. If another program is already using this port, you can change the port number used by the fast launcher.

When a fast launcher option is not used for the amount of time specified in the inactivity timer, it ends, which frees up memory for other applications. By default, the inactivity timer is set to 30 minutes.

To configure the fast launcher option (Interactive SQL)

1. Choose **Tools** » **Options**.
2. In the left pane, click **General**.
3. Click **Configure**.
4. Complete the **Port Number** and **Shut Down The Fast Launcher** fields.
5. Click **OK**.
6. Click **OK**.

To configure the fast launcher option (Sybase Central)

1. Choose **Tools** » **Options**.
2. In the left pane, click **General**.
3. Click **Configure**.
4. Complete the **Port Number** and **Shut Down The Fast Launcher** fields.
5. Click **OK**.
6. Click **OK**.

Changing administration tool language settings

When the fast launcher option is enabled, changes to language settings are only detected by Sybase Central or Interactive SQL once the process is stopped and restarted.

To change the language settings when the fast launcher option is enabled

1. Choose **Tools** » **Options**.
2. On the **General** tab of the **Options** window, clear the **Enable Fast Launcher** option.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Click **OK**.

3. Shut down Sybase Central or Interactive SQL.
4. Change the language settings as required. For example, running the following command changes the language settings to German:

   ```
   dblang DE
   ```

5. Start Sybase Central or Interactive SQL.
6. Re-enable the fast launcher option:
   a. Choose **Tools** » **Options**.
   b. On the **General** tab of the **Options** window, select **Enable Fast Launcher**.
   c. Click **OK**.

To change the language settings when the fast launcher option is disabled

1. Shut down Sybase Central or Interactive SQL.
2. Change the language settings as required. For example, running the following command changes the language settings to German:

   ```
   dblang de
   ```

3. Restart Sybase Central or Interactive SQL.

Alternatively, you can shut down the scjview or dbisql process to stop the fast launcher.

See also

- "Language Selection utility (dblang)" on page 818
- "Using the fast launcher option" on page 757

Using the SQL Anywhere Console utility

The SQL Anywhere Console utility provides administration and monitoring facilities for database server connections.

The SQL Anywhere Console utility is supported on several platforms. For platform availability, see http://www.sybase.com/detail?id=1061806.

On platforms where the SQL Anywhere Console utility is not supported, you can use the connection, database, and database server properties to obtain information or you can monitor your database server from a computer running an operating system that supports the SQL Anywhere Console utility (such as Windows, Mac OS X, or Linux).

If a user without DBA authority connects to the SQL Anywhere Console utility, all features requiring DBA authority are disabled.

For information about the options that the SQL Anywhere Console utility supports, see "SQL Anywhere Console utility (dbconsole)" on page 848.

Starting the SQL Anywhere Console utility

To start the SQL Anywhere Console utility (command line)

- Run the following command:

  ```
  dbconsole
  ```

  If you do not include the -c option, which specifies the connection parameters for the database, or if you supply insufficient connection parameters, the **Connect** window appears, where you can enter connection information for the database.

  For information about the supported options, see "SQL Anywhere Console utility (dbconsole)" on page 848.

  The following command starts the SQL Anywhere Console utility and connects to the sample database:

  ```
  dbconsole -c "UID=DBA;PWD=sql;DSN=SQL Anywhere 12 Demo"
  ```

The following steps can be used if you are using a version of Linux that supports the Linux desktop icons and if you chose to install them when you installed SQL Anywhere.

To start the SQL Anywhere Console utility (Linux desktop icons)

1. From the **Applications** menu, choose **SQL Anywhere 12** » **DBConsole**.
2. Enter the connection information for your database in the **Connect** window.
3. Click **Connect**.

Note
The following steps assume that you have already sourced the SQL Anywhere utilities. See "Setting environment variables on Unix and Mac OS X" on page 377.

To start the SQL Anywhere Console utility (Unix command line)

1. In a terminal session, run the following command:

   ```
   dbconsole
   ```

2. Enter the connection information for your database in the **Connect** window.
3. Click **Connect**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Mac OS X note
The administration tools only run on Intel-based Macintosh computers with 64-bit processors supported by the Apple JDK 1.6 (Mac OS X 10.5.2 or later). See http://www.sybase.com/detail?id=1061806.

To start the SQL Anywhere Console utility (Mac OS X)

1. In the Finder, double-click **DBConsole** in */Applications/SQLAnywhere12*.
2. Enter the connection information for your database in the **Connect** window.
3. Click **Connect**.

See also

- "Working with the Connect window" on page 92

Navigating the SQL Anywhere Console utility main window

The SQL Anywhere Console utility consists of three panes:

- **Connections** Displays information about current database connections.
- **Properties** Displays information about databases and database servers that are currently running.
- **Messages** Displays database server messages.

You can configure the information that appears in each pane using the **Options** window.

To customize the contents of the Connections pane

1. In the SQL Anywhere Console utility, choose **File** » **Options**.
2. In the left pane, click **Connection Viewer**.
3. Select the properties you want to appear in the **Connections** pane.
4. Click **OK**.

To customize the contents of the Properties pane

1. In the SQL Anywhere Console utility, choose **File** » **Options**.
2. In the left pane, click **Property Viewer**.
3. Select the database and database server properties you want to appear in the **Properties** pane.
4. Click **OK**.

To customize the contents of the Messages pane

1. In the SQL Anywhere Console utility, choose **File** » **Options**.
2. In the left pane, click **Message Viewer**.
3. Select the message options for the messages that appear in the **Messages** pane.
4. Click **OK**.

Checking for software updates

You can configure SQL Anywhere to notify you when updates, such as EBFs and maintenance releases, become available. For information about EBFs and maintenance releases, see "-iu option, dbsupport" on page 854.

By default, SQL Anywhere does checks daily for software updates.

Checking for updates automatically

Sybase Central, Interactive SQL, and the SQL Anywhere Console utility (dbconsole) all provide you with a way to configure the Update Checker, which controls whether SQL Anywhere should check for software updates and how often it should do so.

To configure the Update Checker (Sybase Central)

1. Choose **Help** » **SQL Anywhere 12** » **Configure Update Checker**.
2. Edit the Update Checker settings.
3. Click **OK**.

To configure the Update Checker (Interactive SQL)

1. Choose **Tools** » **Options**.
2. In the left pane, click **SQL Anywhere**.
3. Click the **Check For Updates** tab.
4. Edit the Update Checker settings.
5. Click **OK**.

To configure the Update Checker (SQL Anywhere Console utility)

1. Choose **File** » **Options**.
2. In the left pane, click **Check For Updates**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

3. Edit the Update Checker settings.

4. Click **OK**.

Checking for updates manually

You can check for SQL Anywhere software updates at any time by doing one of the following:

- **Start menu** Choose **Start** » **Programs** » **SQL Anywhere 12** » **Check For Updates**.
- **Sybase Central** Choose **Help** » **SQL Anywhere 12** » **Check For Updates**.
- **Interactive SQL** Choose **Help** » **Check For Updates**.
- **SQL Anywhere Console utility (dbconsole)** Choose **Help** » **Check For Updates**.
- **SQL Anywhere Support utility (dbsupport)** Issue the following command:

  ```
  dbsupport -iu
  ```

- **Sybase web site** Go to http://downloads.sybase.com.

See also

- “Error reporting in SQL Anywhere” on page 996
- “Support utility (dbsupport)” on page 853

Database administration utilities

SQL Anywhere includes a set of utility programs for performing database administration tasks. Each of the utilities can be accessed from one or more of Sybase Central, Interactive SQL, or at a command prompt.

For platform availability, see http://www.sybase.com/detail?id=1002288.

The administration utilities use a set of registry entries or *.ini* files. See “Registry and INI files” on page 396.

See also

- “Using Sybase Central” on page 679
- “Using Interactive SQL” on page 697

Using configuration files

Many of the utilities provided with SQL Anywhere 12 allow you to store command-line options in a configuration file. If you use an extensive set of options, you may find it useful to store them in a configuration file.

The *@data* option allows you to specify environment variables and configuration files on the command line. To specify a configuration file, replace *data* with the path and name of the configuration file. If both an environment variable and configuration file exist with the same name, the environment variable is used.

Configuration files can contain line breaks, and can contain any set of options, including the @data option. You can use the number sign (#) to designate lines as comments. The ampersand (&) character appearing by itself at the end of a line indicates that the previous token is continued on the next line. For example, the following configuration file could be used to start a server that allows strong encryption:

```
-ec TLS(TLS_TYPE=RSA;FIPS=Y;IDENTITY=rsaserver.id; &
     IDENTITY_PASSWORD=test)
 -x tcpip c:\mydemo.db
```

The *@data* parameter can occur at any point in the command line, and parameters contained in the file are inserted at that point. You can use *@data* multiple times on one command line to specify multiple configuration files.

Utilities read the command line by expanding the specified configuration files and reading the entire command line from left to right. If you specify options that are overridden by other options in the command line, the option closer to the end of the line wins. Conflicting options can cause errors.

Note
The Start Server in Background utility (dbspawn) does not expand configuration files specified by the *@data* option.

If you want to protect passwords or other information in a configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.

Configuration file escape characters

Within a configuration file, If the value for an option contains values with spaces, then the value must enclosed in double quotes, and the double quotes. Furthermore, if the quoted value contains additional values with spaces, then the double quotes around the additional values must be escaped. SQL Anywhere supports \\ as an escape sequence for a \, and \" as an escape sequence for a ".

For example, this excerpt from a dblsn configuration file has escaped double quote characters:

```
-l "subject=$remote_id;
    content=sync cardealer;
    action='run dbmlsync.exe -c \"filedsn=c:\my fdsns\CarDealer.dsn\"
    -ot dbmlsync.log -k -e sa=on';"
```

Example

The following configuration file holds a set of options for the Validation utility (dbvalid):

```
#Connect to the sample database as the user DBA with password sql
-c "UID=DBA;PWD=sql;DBF=samples-dir\demo.db"
#Perform an express check on each table
-fx
#Log output messages to the specified file
-o "c:\validationlog.txt"
```

For information about *samples-dir*, see "Samples directory" on page 392.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

If this configuration file is saved as *c:\config.txt*, it can be used in a command as follows:

```
dbvalid @c:\config.txt
```

Using conditional parsing in configuration files

You can use conditional parsing in configuration files to specify the utilities that can use the file. Conditional directives allow command parameters to be included or excluded depending on the utility using the file. The File Hiding utility (dbfhide) can still be used to hide the contents of a configuration file when conditional parsing is used in the file.

Syntax

configuration-file= text...

text : *comment* | *conditional* | *command-line-option*

comment : *line starting with # that is not a conditional*

conditional :

#if *condition*
text
[**#elif** *condition*
text
] ...
[**#else**
text
] ...
#endif

condition : { **tool=***utility-name*[,*utility-name*]... | *utility-name* }

The following values are supported for *utility-name*:

dbbackup	dbinfo	dbmlsync	dbstop	dbxtract
dbdsn	dbinit	dbping	dbsupport	mlsrv*X*
dbeng*X*	dblic	dbremote	dbsvc	mluser
dberase	dblocate	dbspawn	dbunload	qaagent
dbfhide	dblog	dbsrv*X*	dbupgrad	
dbhist	dblsn	dbstats	dbvalid	

X is the version number of the executable you want to use. For example, 12 in dbeng12.

Remarks

To be treated as a directive, the first non-whitespace character on a line must be #. When a utility is encountered in an #if or #elif directive, the lines that follow the directive are included until another conditional directive is encountered. The #else directive handles the condition where the utility has not been found in the preceding blocks. The #endif directive completes the conditional directive structure.

Blank spaces are not permitted anywhere within the list of tool names specified by **tool=**. You can nest conditional directives. If an error occurs while parsing the configuration file, the utility reports that the configuration file cannot be opened.

The dbspawn utility allows you to specify a configuration file reference in the command to be spawned, but you cannot specify a configuration file with options for the dbspawn utility. For example, the first command below is supported, but the second command is *not* supported:

```
dbspawn dbeng12 @myconfig.ini

dbspawn @spawnopts.ini dbeng12 demo.db
```

Example

The following configuration file can be used by dbping, dbstop, and dbvalid.

```
#if tool=dbping,dbstop,dbvalid
   #always make tools quiet
   -q
   -c "UID=DBA;PWD=sql;Host=myhost;DBN=mydb"
   #if dbping
      #make a database connection
      -d
   #elif tool=dbstop
      #don't ask
      -y
   #else
      #must be dbvalid
      #use WITH EXPRESS CHECK
      -fx
   #endif
#endif
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Backup utility (dbbackup)" on page 767
- "Data Source utility (dbdsn)" on page 779
- "@data dbeng12/dbsrv12 server option" on page 157
- "Erase utility (dberase)" on page 793
- "File Hiding utility (dbfhide)" on page 794
- "Histogram utility (dbhist)" on page 795
- "Information utility (dbinfo)" on page 797
- "Initialization utility (dbinit)" on page 799
- "Server Licensing utility (dblic)" on page 833
- "Server Enumeration utility (dblocate)" on page 830
- "Transaction Log utility (dblog)" on page 862
- "MobiLink Listener utility for Windows devices (dblsn)" [*MobiLink - Server-Initiated Synchronization*]
- "dbmlsync syntax" [*MobiLink - Client Administration*]
- "Ping utility (dbping)" on page 826
- "SQL Remote Message Agent utility (dbremote)" [*SQL Remote*]
- "Start Server in Background utility (dbspawn)" on page 849
- "Stop Server utility (dbstop)" on page 851
- "Service utility (dbsvc) for Linux" on page 836
- "Service utility (dbsvc) for Windows" on page 840
- "Unload utility (dbunload)" on page 864
- "Upgrade utility (dbupgrad)" on page 879
- "Validation utility (dbvalid)" on page 881
- "Extraction utility (dbxtract)" [*SQL Remote*]
- "mlsrv12 syntax" [*MobiLink - Server Administration*]
- "MobiLink user authentication utility (mluser)" [*MobiLink - Server Administration*]
- "qaagent utility" [*QAnywhere*]

Backup utility (dbbackup)

Creates a client-side or a server-side backup of database files and transaction logs for running databases.

Syntax

dbbackup [*options*] *target-directory*

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.

Option	Description
-b *block-size*	Specifies the maximum block size (in number of pages) to be used to transfer pages from the database server to dbbackup. The dbbackup utility tries to allocate this number of pages; if it fails, it repeatedly reduces this value by half until the allocation succeeds. The default size is 128 pages.
-c *"keyword=value; ..."*	Specifies connection parameters. The user ID must have DBA authority or REMOTE DBA authority to connect to the database. See "Connection parameters" on page 265.
-d	Backs up the main database files only, without backing up the transaction log file, if one exists.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-k *checkpoint-log-copy-option*	Specifies how dbbackup processes the database files before writing them to the destination directory. The choice of whether to apply pre-images during a backup, or copy the checkpoint log as part of the backup, has performance implications. If the -s option is specified to perform the backup on the server, the default setting for -k is auto; otherwise, the default setting is copy. **auto** The database server checks the amount of available disk space on the volume hosting the backup directory. If there is at least twice as much disk space available as the size of the database at the start of the backup, then the backup proceeds as if copy was specified. Otherwise, it proceeds as if nocopy was specified. This setting can only be used if -s is also specified. **copy** The backup reads the database files without applying pre-images for any modified pages. The checkpoint log in its entirety and the system dbspace, are copied to the backup directory. The next time the database is started, the database server automatically recovers the database to its state as of the checkpoint at the start of the backup. Because page pre-images do not have to be written to the temporary file, using this option can provide better backup performance and reduce internal server contention for other connections that are operating during a backup. However, since the backup copy of the database file includes the checkpoint log, which has pre-images of any pages modified since the start of the backup, the backed-up copy of the database files may be larger than the database files at the time the backup started. The copy option should be used when disk space in the destination directory is not an issue. **nocopy** The checkpoint log is not copied as part of the backup. This option causes pre-images of modified pages to be saved in the temporary file so that they can be applied to the backup as it progresses. The backup copies of the database files will be the same size as the database when the backup operation commenced. The backup copies may actually be slightly smaller because the checkpoint log is not present in this copy. This option results in smaller backed up database files, but the backup may proceed more slowly, and possibly decrease performance of other operations in the database server. It is useful in situations where space on the destination drive is limited. **recover** The database server copies the checkpoint log (as with the copy option), but applies the checkpoint log to the database when the backup is complete. This restores the backed up database files to the same state (and size) that they were in at the start of the backup operation. This option is useful if space on the backup drive is limited (it requires the same amount of space as the copy option for backing up the checkpoint log, but the resulting file size is smaller). This setting can only be used if -s is also specified.

Option	Description
-l *filename*	Enables a secondary system to be brought up rapidly in the event of a server crash. A live backup does not stop. It continues running while the server runs. It runs until the primary server becomes unavailable. At that point, it shuts down, but the backed up log file is intact and can be used to bring a secondary system up quickly. See "Differences between live backups and transaction log mirrors" on page 892 and "Make a live backup" on page 900. If you specify -l, then you cannot use -s to create an image back up on the server.
-n	Changes the naming convention of the backup transaction log file to *yymmddxx.log*, where *xx* are sequential letters ranging from **AA** to **ZZ** and *yymmdd* represents the current year, month, and day. This option is used in conjunction with -r. The backup copy of the transaction log file is stored in the directory specified in the command, and with the *yymmddxx.log* naming convention. This allows backups of multiple versions of the transaction log file to be kept in the same backup directory. You can also use both the -x option and the -n option to rename the log copy. For example `dbbackup -c "UID=DBA;PWD=sql" -x -n mybackupdir`
-o *filename*	Writes output messages to the named file.
-p	Sends formatted progress messages from the database server to the client. The -p option is ignored unless the -s option is used to perform a server-side backup. For information about formatted progress messages, see "progress_messages option" on page 576.
-q	Suppresses output messages. This option is available only when you run this utility from a command prompt.
-r	Renames the transaction log and starts a new transaction log. It forces a checkpoint and causes the following three steps to occur: 1. The current working transaction log file is copied and saved to the directory specified in the command. 2. The current transaction log remains in its current directory, but is renamed using the format *yymmddxx.log*, where *xx* are sequential characters starting at *AA* and running through to *ZZ*, and *yymmdd* represents the current year, month, and day. This file is then no longer the current transaction log. 3. A new transaction log file is generated that contains no transactions. It is given the name of the file that was previously considered the current transaction log, and is used by the database server as the current transaction log.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-s	Creates an image backup on the server using the BACKUP DATABASE statement. If you specify the -s option, the -l option (to create a live backup of the transaction log) cannot be used. The directory specified is relative to the server's current directory, so it is recommended that you specify a full pathname. In addition, the server must have write permissions on the specified directory. When -s is specified, the Backup utility does not display progress messages and does not prompt you when it overwrites existing files. If you want to be prompted when an attempt is made to overwrite an existing file, do not specify -s or -y. You must specify -s if you specify the -k recovery option.
-t	Creates a backup that can be used as an incremental backup since the transaction log can be applied to the most recently backed up copy of the database file(s).
-x	Backs up the existing transaction log, deletes the original log, and then starts a new transaction log. Do not use this option if you are using database mirroring. See "Database mirroring and transaction log files" on page 971.
-xo	Deletes the current transaction log and starts a new one. This operation does not perform a backup; its purpose is to free up disk space in non-replication environments. Do not use this option if you are using database mirroring. See "Database mirroring and transaction log files" on page 971. **Caution** Do not use the -xo option with databases that are being replicated or synchronized. SQL Remote and MobiLink rely on transaction log information.
-y	Creates the backup directory or replaces a previous backup file in the directory without confirmation. If you want to be prompted when an attempt is made to overwrite an existing file, do not specify -s or -y.
target-directory	Specifies the directory the backup files are copied to. If the directory does not exist, it is created. However, the parent directory must exist. By default, the Backup utility creates a client-side backup of the database files. You can specify -s to create a backup on the server using the BACKUP DATABASE statement.

Remarks

The Backup utility makes a backup copy of all the files for a single database. A simple database consists of two files: the main database file and the transaction log. More complicated databases can store tables in multiple files, with each file as a separate dbspace. All backup file names are the same as the database file names. The image backup created by the Backup utility consists of a separate file for each file that is backed up.

For more information about making archive backups (a single file that contains both the database file and the transaction log), see "Archive backups" on page 892.

Using the Backup utility on a running database is equivalent to copying the database files when the database is not running. You can use the Backup utility to back up the database while other applications or users are using it.

If neither of the options -d or -t are used, all database files are backed up.

By default, the Backup utility creates a client-side backup of the database files. You can specify -s to create a backup on the server using the BACKUP DATABASE statement.

For information about performing server-side backups, see "BACKUP statement" [*SQL Anywhere Server - SQL Reference*].

Caution
Backup copies of the database and transaction log must not be changed in any way. If there were no transactions in progress during the backup, or if you specified BACKUP DATABASE WITH CHECKPOINT LOG RECOVER or WITH CHECKPOINT LOG NO COPY, you can check the validity of the backup database using read-only mode or by validating a copy of the backup database.

However, if transactions were in progress, or if you specified BACKUP DATABASE WITH CHECKPOINT LOG COPY, the database server must perform recovery on the database when you start it. Recovery modifies the backup copy, which is not desirable.

In addition to dbbackup, you can access the Backup utility in the following ways:

- From Sybase Central, using the **Create Backup Images Wizard**. See "Image backups" on page 893.
- From Interactive SQL, using the BACKUP DATABASE statement. See "BACKUP statement" [*SQL Anywhere Server - SQL Reference*].

For more information about recommended backup procedures, see "Backup and data recovery" on page 887.

Exit codes are 0 (success) or non-zero (failure).

For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

Example

For example, the following command backs up the sample database running on the computer named sample_host, connecting as the DBA user, into the *SQLAnybackup* directory:

```
dbbackup -c "Host=sample_host;DBN=demo;UID=DBA;PWD=sql" SQLAnybackup
```

Broadcast Repeater utility (dbns12)

Allows SQL Anywhere clients to find SQL Anywhere database servers running on other subnets and through firewalls where UDP broadcasts normally do not reach.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Syntax

dbns12 [*options*] [*address* ...]

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-ap *port*	Specifies the port number used by the database server. The default port number is 2638.
-m *ip*	Specifies the IP address of the computer the DBNS process is running on. This parameter is required for computers with more than one IP address. This address must be an IPv4 address.
-o *filename*	Writes the output that appears in the Broadcast Repeater messages window to the named file.
-p *port*	Specifies the port number used by the DBNS Broadcast Repeater. The default is 3968. If there is a firewall between the subnets, then you must open the port number used by the Broadcast Repeater utility for TCP connections between DBNS processes, in addition to opening port 2638 for standard client-server communications.
-q	Runs in quiet mode—messages are not displayed.
-s	Causes the new DBNS process to check if another DBNS process is already running on that subnet, and returns an error before shutting down if another DBNS process is found.
-ud	Runs the DBNS Broadcast Repeater as a daemon on Unix operating systems. If you run the DBNS Broadcast Repeater as a daemon, it is recommended that you specify the -o option to log output information. When you start the DBNS Broadcast Repeater as a daemon, its permissions are controlled by the current user's umask setting. It is recommended that you set the umask value before starting the DBNS Broadcast Repeater to ensure that it has the appropriate permissions.
-ui	For Linux with X window server support, starts the DBNS Broadcast Repeater in shell mode if a usable display is not available.
-ux	Opens the **SQL Anywhere Broadcast Repeater** window on Linux (use the X window server).
-x *host*	Shuts down the DBNS process running on specified host. You can specify an IP address or host name.

Option	Description
-z	Starts the DBNS Broadcast Repeater in debug mode. When running in debug mode, a line appears in the Broadcast Repeater messages window for each SQL Anywhere broadcast packet that is received or forwarded. Debug mode should only be used when there are connectivity problems because of the verbosity of the debugging output.
address	Specifies the IP address or host name of other computers that are, or will be, running DBNS processes. This allows the DBNS processes to detect each other and exchange information about known database servers and other DBNS processes.

Remarks

The Broadcast Repeater allows SQL Anywhere clients to find SQL Anywhere database servers running on other subnets and through firewalls where UDP broadcasts normally do not reach, without using the HOST connection parameter or LDAP.

The *address* can be either an IP address or a computer name. Use spaces to separate multiple addresses.

This utility is available on supported Unix and all 32-bit and 64-bit Windows platforms.

The clients and database server must be running SQL Anywhere 9.0.2 or later to use the Broadcast Repeater.

Caution
It is recommended that you do not run the dbns12 utility on the same computer as a SQL Anywhere database server because dbns12 or the database server may not receive UDP broadcasts.

See also

- "Locating a database server using the Broadcast Repeater utility" on page 143

Example

Suppose you want to allow computers on the subnets 10.50.83.255 and 10.50.125.255 to connect using broadcasts. You need to a computer on the 10.50.83.255 subnet (Computer A at 10.50.83.114) and one computer on the 10.50.125.255 subnet (Computer B at 10.50.125.103).

On each of these two computers, run dbns12, passing the IP address of the other computer. Execute the following command on Computer A:

```
dbns12 10.50.125.103
```

On Computer B, execute the following command:

```
dbns12 10.50.83.114
```

If either computer has more than one IP address, you must also specify the local IP address using the -m option. For example, on Computer A, you would use the following command:

```
dbns12 -m 10.50.83.114  10.50.125.103
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Certificate Creation utility (createcert)

Creates X.509 certificates.

Syntax

createcert [**-r** | **-s**]

Option	Description
-r	Creates a PKCS10 certificate request. When this option is specified, createcert does not prompt for a signer or any other information used to sign a certificate.
-s *filename*	Signs the PKCS10 certificate request that is in the specified file. The request can be DER or PEM encoded. When this option is specified, createcert does not prompt for key generation or subject information.

Remarks

Users may typically go to a third party to purchase certificates. These certificate authorities provide their own tools for creating certificates. The following tools may be especially useful to create certificates for development and testing purposes, and can also be used for production certificates.

To create a signed certificate, use createcert without options. If you want to break up the process into two steps, for example so one person creates a request and another person signs it, the first person can run createcert with -r to create a request and the second person can sign the request by running createcert with -s.

When you run createcert, you are prompted for the following information. When you specify the -r or -s option, some of these prompts do not appear.

- **Choose encryption type** This prompt only appears if you have purchased a license for ECC encryption. Choose RSA or ECC.
- **Enter RSA key length (512-16384)** This prompt only appears if you chose RSA encryption. You can choose a length between 512 bits and 16384 bits.
- **Enter ECC curve** This prompt only appears if you have purchased a license for ECC encryption and you chose the ECC encryption type above. You are prompted to choose from a list of ECC curves. The default is sect163k1.
- **Subject information** You must enter the following information, which identifies the entity:
  - Country Code
  - State/Province
  - Locality
  - Organization
  - Organizational Unit
  - Common Name

- **Enter file path of signer's certificate** Optionally, supply a location and file name for the signer's certificate. If you supply this information, the generated certificate is a signed certificate. If you do not supply this information, then the generated certificate is a self-signed root certificate.
- **Enter file path of signer's private key** Supply a location and file name to save the private key associated with the certificate request. This prompt only appears if you supplied a file in the previous prompt.
- **Enter password for signer's private key** Supply the password that was used to encrypt the signer's private key. Only supply this password if the private key was encrypted.
- **Serial number** Optionally, supply a serial number. The serial number must be a hexadecimal string of 40 digits or less. This number must be unique among all certificates signed by the current signer. If you do not supply a serial number, createcert generates a GUID as the serial number.
- **Certificate will be valid for how many years (1-100)** Specify the number of years (between 1 and 100) that the certificate is valid. After this period, the certificate expires, along with all certificates it signs.
- **Certificate Authority (y)es or (n)o** Indicate whether this certificate can be used to sign other certificates. By default, certificates are not certificate authorities (n).
- **Key usage** Supply a comma-separated list of numbers that indicate how the certificate's private key can be used. This is an advanced option; the default should be acceptable for most situations. The default depends on whether the certificate is a certificate authority or not.
- **File path to save request** This prompt only appears if you specify the -r option. Supply a location and file name for the PCKS10 certificate request.
- **Enter file path to save certificate** Supply a location and file name to save the certificate. The certificate is not saved unless you specify a location and file name.
- **Enter file path to save private key** Supply a location and file name to save the private key.
- **Enter password to protect private key** Optionally, supply a password with which to encrypt the private key. If you do not supply a password, the private key is not encrypted. This prompt only appears if you supplied a file in the previous prompt.
- **Enter file path to save identity** Supply a location and file name to save the identity. The identity file is a concatenation of the certificate, signer, and private key. This is the file that you supply to the server at startup. If the private key was not saved, createcert prompts for a password to save the private key. Otherwise, it uses the password provided earlier. The identity is not saved unless you provide a file name. If you do not save the identity file, you can manually concatenate the certificate, signer, and private key files into an identity file.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- “Certificates” on page 1144
- “Certificate Viewer utility (viewcert)” on page 778
- “-ec dbeng12/dbsrv12 server option” on page 176
- “Encryption (ENC) connection parameter” on page 287
- “FIPS-approved encryption technology” on page 1144

Example

The following example creates a signed certificate. In the example, no file name is provided for the signer's certificate, which makes it a self-signed root certificate.

```
>createcert
SQL Anywhere X.509 Certificate Generator Version 12.0.0.3330
Choose encryption type ((R)SA or (E)CC): r
Enter RSA key length (512-16384): 1024
Generating key pair...
Country Code: CA
State/Province: Ontario
Locality: Waterloo
Organization: Sybase iAnywhere
Organizational Unit: Engineering
Common Name: Test Certificate
Enter file path of signer's certificate:
Certificate will be a self-signed root
Serial number [generate GUID]:
Generated serial number: bfb89a26fb854955954cabc4d056e177
Certificate valid for how many years (1-100): 10
Certificate Authority (Y/N) [N]: n
1.  Digital Signature
2.  Nonrepudiation
3.  Key Encipherment
4.  Data Encipherment
5.  Key Agreement
6.  Certificate Signing
7.  CRL Signing
8.  Encipher Only
9.  Decipher Only
Key Usage [3,4,5]: 3,4,5
Enter file path to save certificate: cert.pem
Enter file path to save private key: key.pem
Enter password to protect private key: pwd
Enter file path to save identity: id.pem
```

To generate an enterprise root certificate (a certificate that signs other certificates), a self-signed root certificate should be created with Certificate Authority. The procedure is similar to that shown above. However, the response to the Certificate Authority prompt should be **yes** and choice for roles should be option **6,7** (the default).

```
Certificate Authority (Y/N) [N]: y
1.  Digital Signature
2.  Nonrepudiation
3.  Key Encipherment
4.  Data Encipherment
5.  Key Agreement
6.  Certificate Signing
7.  CRL Signing
8.  Encipher Only
```

```
9.  Decipher Only
Key Usage [6,7]: 6,7
```

Certificate Viewer utility (viewcert)

Displays values within a Public Key Infrastructure (PKI) object, converts the encoding of PKI objects, or encrypts and decrypts private keys.

Syntax

viewcert [*options*] *input-file*

Option	Description
-d	DER-encodes the output. This option is only useful with the -o option. It cannot be used with -p. By default, viewcert outputs the PKI object in a readable text format.
-ip *input-password*	Specifies the password needed to decrypt the private key if the *input-file* contains an encrypted private key.
-o *output-file*	Specifies the file that viewcert should write the output to. By default, viewcert writes the output to the command prompt window where it is running.
-op *output-password*	Specifies the password viewcert should use to encrypt a private key. This option is only useful with -d or -p. By default, private keys are not encrypted.
-p	PEM-encodes the output. This option is only useful with the -o option. It cannot be used with -d. By default, viewcert outputs the PKI object in a readable text format.
input-file	Specifies a file that must be a DER- or PEM-encoded PKI object.

Remarks

The viewcert utility can be used to view the following types of PKI objects:

- X.509 certificates
- certificate requests
- private keys
- certificate revocation lists (CRLs)

Viewcert can also be used to convert between DER and PEM encoding types and to encrypt or decrypt private keys.

The viewcert utility supports RSA and ECC objects. To view ECC objects, you must order a separate license. See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Certificate Creation utility (createcert)" on page 775

Example

The following example allows you to view the sample RSA certificate that is included with SQL Anywhere:

```
viewcert rsaroot.crt
```

This example produces the following output:

```
SQL Anywhere X.509 Certificate Viewer Version 12.0.0.2413

X.509 Certificate
-----------------
Common Name:              RSA Root
Country Code:             test
State/Province:           test
Locality:                 test
Organization:             test
Organizational Unit:      test
Issuer:                   RSA Root
Serial Number:            303031
Issued:                   Apr 15, 2002  12:53:51
Expires:                  Apr 16, 2022  12:53:51
Signature Algorithm:      RSA, MD5
Key Type:                 RSA
Key Size:                 1024 bits
Basic Constraints:        Is a certificate authority, path length limit: 10
Key Usage:                Certificate Signing, CRL Signing
```

Data Source utility (dbdsn)

Creates, deletes, describes, and lists SQL Anywhere ODBC data sources.

Syntax

dbdsn [*modifier-options*]
{ **-l**[**s** | **u**]
| **-d**[**s** | **u**] *dsn*
| **-g**[**s** | **u**] *dsn*
| **-w**[**s** | **u**] *dsn* [*details-options*;...]
| **-cl** }

Major option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.

Major option	Description
-l[**s** | **u**]	Lists the available SQL Anywhere ODBC data sources. You can modify the list format using the -b or -v options. On Windows, you can modify the option using the **u** (user) or **s** (system) specifiers. The default specifier is **u**.
-d[**s** | **u**] *dsn*	Deletes the named SQL Anywhere data source. If you supply -y, any existing data source is deleted without confirmation. On Windows, you can modify the option using the **u** (user) or **s** (system) specifiers. The default specifier is **u**.
-g[**s** | **u**] *dsn*	Lists the definition of the named SQL Anywhere data source. You can modify the format of the output using the -b or -v options. On Windows, you can modify the option using the **u** (user) or **s** (system) specifiers. The default specifier is **u**.
-w[**s** | **u**] *dsn* [*details-options*]	Creates a new data source, or overwrites one if one of the same name exists. You must specify the -c option with the -w option. If you supply -y, any existing data source is overwritten without confirmation. On Windows, you can modify the option using the **u** (user) or **s** (system) specifiers. The default specifier is **u**.
-cl	Lists the connection parameters supported by the dbdsn utility. See "Connection parameters" on page 265. For information about supported ODBC connection parameters, see "ODBC connection parameters" on page 785.

Modifier-options	Description
-b	Formats the output of the list as a single line connection string.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Modifier-options	Description
-cm	Displays the command used to create the data source. This option can be used to output the creation command to a file, which can be used to add the data source to another computer or can be used to restore a data source to its original state if changes have been made to it. You must specify the -g option or -l option with -cm or the command fails. Specifying -g displays the creation command for the specified data source, while specifying -l displays the creation command for all data sources. If the specified data source does not exist, the command to delete the data source is generated. For example, if the mydsn data source does not exist on the computer, `dbdsn -cm -g mydsn` would return the following command to delete the mydsn data source: `dbdsn -y -du "mydsn"`
-dr	Includes the Driver parameter when displaying data sources. When you use the -cm option to recreate data sources, it allows the current version of dbdsn to create data sources that reference a different version of the ODBC driver. For example, suppose you used the following command to create a version 9 data source: `dbdsn -y -wu "9.0 Student Sample" -c "UID=DBA;PWD=sql;...;Driver= Adaptive Server Anywhere 9.0"` When you execute `dbdsn -cm -l`, dbdsn lists the same command without the Driver= parameter, which would then recreate the ODBC data source using the SQL Anywhere version 10.0 ODBC driver. However, if you execute `dbdsn -dr -cm -l`, then the Driver= parameter is included and the data source is recreated exactly as it was created originally: using the version 9 ODBC driver.
-f	Displays the name of the system file that is being used. This option is only available on Unix.
-ns	Specifies that the environment variable settings are used to determine the location of the system information file (named *.odbc.ini* by default). This option is also useful for determining which file is being used by dbdsn when there are multiple candidates for the system information file in the environment. This option is only available on Unix. If you do not specify -ns when creating a data source, dbdsn also checks for the system information file in the user's home directory and the path. For more information about how the system information file is located, see "Using ODBC data sources on Unix" on page 105.

Modifier-options	Description
-o *filename*	Writes output messages to the named file.
-or	Creates a data source for the iAnywhere Solutions Oracle driver when specified with the -c option. For example: `dbdsn -w MyOracleDSN -or -c Userid=DBA;Password=sql; ServiceName=abcd;ArraySize=500;ProcResults=y` You can specify the -cl option with the -or option to obtain a list of the connection parameters for the iAnywhere Solutions Oracle driver. For more information, see "iAnywhere Solutions 12 - Oracle ODBC driver" [*MobiLink - Server Administration*].
-pe	Encrypts the password specified in the Password connection parameter, and stores the encrypted password in the data source using the ENP connection parameter.
-q	Suppresses output to the database server messages window. If you specify -q when deleting or modifying a data source, you must also specify -y.
-v	Formats the output of the list over several lines, as a table.
-y	Deletes or overwrites each data source without prompting you for confirmation. If you specify -q when deleting or modifying a data source, you must also specify -y.

Details-options	Description
-c "*keyword=value*;..."	Specifies connection parameters as a connection string. See "Connection parameters" on page 265.
-cw	Ensures that the DBF parameter (specified using -c) is an absolute file name. If the value of DBF is not an absolute file name, dbdsn will prepend the current working directory (CWD). This option is useful because some operating systems do not have CWD information readily available in batch files.

Remarks

The modifier options can occur before or after the major option specification.

The Data Source utility is a cross-platform alternative to the ODBC Data Source Administrator for creating, deleting, describing, and listing SQL Anywhere ODBC data sources. The utility is useful for batch operations.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Caution
Storing user IDs, passwords (encrypted or unencrypted), and/or database keys in a data source is not secure. It is recommended that you do not store this information in a data source if the database contains sensitive data.

On Windows operating systems, the data sources are held in the registry.

For information about creating a data source on Windows using the ODBC Data Source Administrator, see "Creating ODBC data sources" on page 98.

On Unix operating systems, data sources are held in the system information file (named *.odbc.ini* by default). When you use the Data Source utility to create or delete SQL Anywhere ODBC data sources on Unix, the utility automatically updates the `[ODBC Data Sources]` section of the system information file. If you do not specify the Driver connection parameter using the -c option on Unix, the Data Source utility automatically adds a Driver entry with the full path of the SQL Anywhere ODBC driver based on the setting of the SQLANY12 environment variable.

For more information about the system information file, see "Using ODBC data sources on Unix" on page 105.

For information about supported ODBC connection parameters, see "ODBC connection parameters" on page 785.

Caution
You should not obfuscate the system information file (*.odbc.ini*) with the File Hiding utility (dbfhide) on Unix unless you will only be using SQL Anywhere data sources. If you plan to use other data sources (for example, for MobiLink synchronization), then obfuscating the system information file may prevent other drivers from functioning properly.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

See also

- "Creating ODBC data sources" on page 98
- "Using ODBC data sources on Unix" on page 105

Example

Write a definition of the data source newdsn. Do not prompt for confirmation if the data source already exists.

```
dbdsn -y -w newdsn -c "UID=DBA;PWD=sql;Host=myhost"
```

or, with a different option order,

```
dbdsn -w newdsn -c "UID=DBA;PWD=sql;Host=myhost" -y
```

List all known user data sources, one data source name per line:

```
dbdsn -l
```

List all known system data sources, one data source name per line:

```
dbdsn -ls
```

List all data sources along with their associated connection string:

```
dbdsn -l -b
```

Report the connection string for the user data source MyDSN:

```
dbdsn -g MyDSN
```

Report the connection string for the system data source MyDSN:

```
dbdsn -gs MyDSN
```

Delete the data source BadDSN, but first list the connection parameters for BadDSN and prompt for confirmation:

```
dbdsn -d BadDSN -v
```

Delete the data source BadDSN without prompting for confirmation.

```
dbdsn -d BadDSN -y
```

Create a data source named NewDSN for the database server MyServer:

```
dbdsn -w NewDSN -c "UID=DBA;PWD=sql;Server=MyServer"
```

If NewDSN already exists, you are prompted to confirm overwriting the data source.

List all connection parameter names and their aliases:

```
dbdsn -cl
```

List all user data sources:

```
dbdsn -l -o dsninfo.txt
```

List all connection parameter names:

```
dbdsn -cl -o dsninfo.txt
```

Specify an absolute file name. When the ODBC data source is created, it contains *DBF=c:\SQLAnywhere12\my.db*.

```
c:\SQLAnywhere12> dbdsn -w testdsn -cw -c
UID=DBA;PWD=sql;Server=SQLAny;DBF=my.db
```

Generate the command to create the SQL Anywhere 12 Demo data source and output it to a file called *restoredsn.bat*:

```
dbdsn -cm -gs "SQL Anywhere 12 Demo" > restoredsn.bat
```

The *restoredsn.bat* file contains the following:

```
dbdsn -y -ws "SQL Anywhere 12 Demo" -c "UID=DBA;PWD=sql;
DBF='C:\Documents and Settings\All Users\Documents\SQL Anywhere 12\Samples
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
\demo.db';
Server=demo12;START='C:\Program Files\SQL Anywhere 12\Bin32\dbeng12.exe';
ASTOP=yes;Description='SQL Anywhere 12 Sample Database'"
```

Return the location of the system information file on Unix:

```
dbdsn -f
```

This command returns the following output:

```
dbdsn using /home/user/.odbc.ini
```

Change the location of the system information file:

```
export ODBCINI=./myodbc.ini
```

Verify the new location of the system information file using `dbdsn -f`:

```
dbdsn using ./myodbc.ini
```

Use the -ns option when creating the data source:

```
dbdsn -w NewDSN -c "UID=DBA" -ns
```

This results in the following output:

```
Configuration "newdsn" written to file ./myodbc.ini
```

ODBC connection parameters

The Data Source utility (dbdsn) supports the following ODBC connection parameters. Boolean (true or false) arguments are either YES or 1 if true, or NO or 0 if false.

Name	Description
Delphi	Delphi cannot handle multiple bookmark values for a row. When you set this value to NO, one bookmark value is assigned to each row, instead of the two that are otherwise assigned. Setting this option to YES can improve scrollable cursor performance.

Name	Description
DescribeCursor	This parameter lets you specify how often you want a cursor to be redescribed when a procedure is executed. The default setting is If Required. • **Never** Specify 0, N, or NO if you know that your cursors do not have to be redescribed. Redescribing cursors is expensive and can decrease performance. • **If Required** Specify 1, Y, or YES if you want the ODBC driver to determine whether a cursor must be redescribed. The presence of a RESULT clause in your procedure prevents ODBC applications from redescribing the result set after a cursor is opened. This is the default setting. • **Always** If you specify 2, A, or ALWAYS, the cursor is redescribed each time it is opened. If you use Transact-SQL procedures or procedures that return multiple result sets, you must redescribe the cursor each time it is opened.
Description	This parameter allows you to provide a description of the ODBC data source.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Name	Description
Driver	This parameter allows you to specify an ODBC driver for the connection, as follows: `Driver=`*`driver-name`*. By default, the driver that is used is **SQL Anywhere 12**. The *driver-name* must be SQL Anywhere *X*, where *X* is the major version number of the software. If the *driver-name* does not begin with **SQL Anywhere**, it cannot be read by the Data Source utility (dbdsn). On Unix, this parameter specifies the fully-qualified path to the shared object. If you do not specify the Driver connection parameter on Unix, the Data Source utility automatically adds a Driver entry with the full path of the SQL Anywhere ODBC driver based on the setting of the SQLANY12 environment variable.
Escape	This parameter specifies the escape character used in the LIKE clause of SQL statements that are generated by the ODBC driver when returning a list of tables or columns. By default the ODBC driver uses the tilde character (~), but some applications assume that the escape character is the backslash character (\). The following connection string fragment specifies that the backslash is the escape character: `"DSN=SQL Anywhere 12 Demo;UID=DBA;PWD=sql;ESCAPE=\\;Host=myhost"`
GetTypeInfoChar	When this option is set to YES, CHAR columns are returned as SQL_CHAR instead of SQL_VARCHAR. By default, CHAR columns are returned as SQL_VARCHAR.

Name	Description
InitString	InitString allows you to specify a command that is executed immediately after the connection is established. For example, you may want to set a database option or execute a stored procedure.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Name	Description
IsolationLevel	You can specify one of the following values to set the initial isolation level for this data source: • **0** This is also called the read uncommitted isolation level. This is the default isolation level. It provides the maximum level of concurrency, but dirty reads, non-repeatable reads, and phantom rows may be observed in result sets. • **1** This is also called the read committed level. This provides less concurrency than level 0, but eliminates some of the inconsistencies in result sets at level 0. Non-repeatable rows and phantom rows may occur, but dirty reads are prevented. • **2** This is also called the repeatable read level. Phantom rows may occur. Dirty reads and non-repeatable rows are prevented. • **3** This is also called the serializable level. This provides the least concurrency, and is the strictest isolation level. Dirty reads, non-repeatable reads, and phantom rows are prevented. • **Snapshot** You must enable snapshot isolation for the database to use this isolation level. The snapshot isolation levels prevent all interference between reads and writes. Writes can still interfere with each other. For contention, a few inconsistencies are possible and performance is the same as isolation level 0. • **Statement-snapshot** You must enable snapshot isolation for the database to use this isolation level. The snapshot isolation levels prevent all interference between reads and writes. Writes can still interfere with each oth-

Name	Description
	er. For contention, a few inconsistencies are possible and performance is the same as isolation level 0. • **Readonly-statement-snapshot** You must enable snapshot isolation for the database to use this isolation level. The snapshot isolation levels prevent all interference between reads and writes. Writes can still interfere with each other. For contention, a few inconsistencies are possible and performance is the same as isolation level 0. For more information, see "Choosing isolation levels" [*SQL Anywhere Server - SQL Usage*].
KeysInSQLStatistics	Set this parameter to YES if you want the SQLStatistics function to return foreign keys. The ODBC specification states that SQLStatistics should not return primary and foreign keys; however, some Microsoft applications (such as Microsoft Visual Basic and Microsoft Access) assume that primary and foreign keys are returned by SQLStatistics.
LazyAutocommit	Set this parameter to YES to delay the commit operation until a statement closes.
PrefetchOnOpen	When PrefetchOnOpen is set to YES, a prefetch request is sent with a cursor open request. The prefetch eliminates a network request to fetch rows each time a cursor is opened. Columns must already be bound for the prefetch to occur on the open. This connection parameter can help reduce the number of client/server requests to help improve performance over a LAN or WAN.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Name	Description
PreventNotCapable	The SQL Anywhere ODBC driver returns a `Driver not capable` error because it does not support qualifiers. Some ODBC applications do not handle this error properly. Set this parameter to YES to prevent this error code from being returned, allowing these applications to work.
SuppressWarnings	Set this parameter to YES if you want to suppress warning messages that are returned from the database server on a fetch. Versions 8.0.0 and later of the database server return a wider range of fetch warnings than earlier versions of the software. For applications that are deployed with an earlier version of the software, you can select this option to ensure that fetch warnings are handled properly.
TranslationDLL	This option is provided for backward compatibility. The use of translators is not recommended.
TranslationName	This option is provided for backward compatibility. The use of translators is not recommended.
TranslationOption	This option is provided for backward compatibility. The use of translators is not recommended.

dbisqlc utility (deprecated)

The dbisqlc utility executes SQL statements against a database.

Note
The dbisqlc utility does not support all the SQL statements and features that Interactive SQL supports and may not support all the features available in the current version of the database server. It is recommended that you use the Interactive SQL utility. See "Interactive SQL utility (dbisql)" on page 812.

Syntax

dbisqlc [*options*] [*dbisqlc-command* | *command-file*]

Option	Description
-c *"keyword=value; ..."*	Specifies connection parameters. If Interactive SQL cannot connect, you are presented with a window where you can enter the connection parameters. See "Connection parameters" on page 265.
-d *delimiter*	Specifies a command delimiter. Quotation marks around the delimiter are optional, but are required when the command shell itself interprets the delimiter in some special way. The specified command delimiter is used for all connections in the current dbisqlc session.
-q	Suppresses output messages. This is useful only if you start Interactive SQL with a command or command file. Specifying this option does not suppress error messages, but it does suppress the following: • warnings and other non-fatal messages • the printing of result sets
-x	Scans commands but does not execute them. This is useful for checking long command files for syntax errors.

Remarks

The dbisqlc utility allows you to type SQL commands or run command files.

If *dbisqlc-command* is specified, dbisqlc executes the command. You can also specify a command file name. If no *dbisqlc-command* or *command-file* argument is specified, dbisqlc enters interactive mode, where you can type a command into a command window.

The dbisqlc utility is available on Microsoft Windows, Mac OS X, and Unix.

See also

- "Interactive SQL utility (dbisql)" on page 812
- "SQL language elements" [*SQL Anywhere Server - SQL Reference*]

Example

The following command runs the command file *mycom.sql* against the current default server, using the user ID DBA and the password sql. If there is an error in the command file, the process shuts down.

```
dbisqlc -c "UID=DBA;PWD=sql" mycom.sql
```

The following command adds a user to the current default database:

```
dbisqlc -c "UID=DBA;PWD=sql" CREATE USER joe IDENTIFIED BY passwd
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Erase utility (dberase)

Erases dbspaces and transaction log files associated with a database.

Syntax

dberase [*options*] *database-file*

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-ek *key*	Specifies the encryption key for strongly encrypted databases directly in the command. If you have a strongly encrypted database, you must provide the encryption key to use the database or transaction log in any way. For strongly encrypted databases, you must specify either -ek or -ep, but not both. The command will fail if you do not specify the correct key for a strongly encrypted database.
-ep	Specifies that you want to be prompted for the encryption key. This option causes a window to appear, in which you enter the encryption key. It provides an extra measure of security by never allowing the encryption key to be seen in clear text. For strongly encrypted databases, you must specify either -ek or -ep, but not both. The command will fail if you do not specify the correct key for a strongly encrypted database.
-o *filename*	Writes output messages to the named file.
-q	Runs in quiet mode—messages are not displayed. If you specify this option, you must also specify -y, otherwise the operation fails.
-y	Deletes each file without being prompted for confirmation. If you specify -q, you must also specify -y, otherwise the operation fails.

Remarks

With the Erase utility, you can erase a database file and its associated transaction log, or you can erase a transaction log file or transaction log mirror file. All database files and transaction log files are marked read-only to prevent accidental damage to the database and accidental deletion of the database files.

The *database-file* may be a database file or transaction log file. The full file name must be specified, including extension. If a database file is specified, the associated transaction log file (and mirror, if one is maintained) is also erased.

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Note
The Erase utility does not erase dbspaces. If you want to erase a dbspace, use DROP DATABASE statement or the **Erase Database Wizard** in Sybase Central. See "DROP DBSPACE statement" [*SQL Anywhere Server - SQL Reference*].

You can also use the **Erase Database Wizard** to erase dbspaces and transaction log files. See "Erasing a database" on page 31.

Deleting a database file that references other dbspaces does not automatically delete the dbspace files. If you want to delete the dbspace files on your own, change the files from read-only to writable, and then delete the files individually. As an alternative, you can use the DROP DATABASE statement to erase a database and its associated dbspace files.

If you erase a database file, the associated transaction log and transaction log mirror are also deleted. If you erase a transaction log for a database that also maintains a transaction log mirror, the mirror is not deleted.

The database being erased must not be running when this utility is used.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

File Hiding utility (dbfhide)

Uses simple encryption to hide the contents of configuration files and initialization files.

Syntax

dbfhide [*options*]*original-configuration-file encrypted-configuration-file*

Option	Description
-q	Runs in quiet mode—messages are not displayed.
-w	Encrypts the file with an encryption API supplied by Microsoft. The file can only be used by SQL Anywhere database servers or tools on the same computer on which the file was encrypted. Only the user who encrypted the file can use the file. This option is only supported on Windows.
-wm	Encrypts the file with an encryption API supplied by Microsoft. The file can only be used by SQL Anywhere database servers or tools on the same computer on which the file was encrypted. Any user on the computer where the file was encrypted can read the file. This option is only supported on Windows.
original-configuration-file	Specifies the name of the original file.
encrypted-configuration-file	Specifies a name for the new obfuscated file.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

Configuration files are used by some utilities to hold command line options. These options may contain a password. You can use the File Hiding utility to add simple encryption to configuration files, and to *.ini* files used by SQL Anywhere and its utilities, and thereby obfuscate the contents of the file. The original file will not be modified. Once you add simple encryption to a file, there is no way to remove it. To make changes to an obfuscated file, you must keep a copy of the original file that you can modify and obfuscate again.

For more information about using configuration files, see "Using configuration files to store database server startup options" on page 41.

For more information about encryption, see "Keeping your data secure" on page 1115.

This utility does not accept the ***@data*** parameter to read in options from a configuration file.

See also

- "Using configuration files" on page 763
- "Using conditional parsing in configuration files" on page 765
- "Hiding the contents of .ini files" on page 397

Example

Create a configuration file that starts the personal database server and the sample database. It should set a cache of 10 MB, and name this instance of the personal server *Elora*. The configuration file would be written as follows:

```
# Configuration file for server Elora
-n Elora
-c 10M
samples-dir\demo.db
```

(Note that lines beginning with # are treated as comments.)

For information about *samples-dir*, see "Samples directory" on page 392.

Name the file *sample.txt*. If you wanted to start the database using this configuration file, your command line would be:

```
dbeng12 @sample.txt
```

Now, add simple encryption to the configuration.

```
dbfhide sample.txt encrypted_sample.txt
```

Use the encrypted_sample.txt file to start a database.

```
dbsrv12 @encrypted_sample.txt
```

Histogram utility (dbhist)

Converts a histogram into a Microsoft Excel chart containing information about the selectivity of predicates.

Syntax

dbhist [*options*] **-t** *table-name* [*excel-output-filename*]

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-c *options*	Specifies connection parameters. See "Connection parameters" on page 265.
-n *colname*	Specifies the name of the column to associate the histogram with. If you do not specify a column, all columns that have histograms in the table are returned.
-t *table-name*	Specifies the name of the table or materialized view for which to generate the chart.
-u *owner*	Specifies the owner of the table or materialized view.
excel-output-name	Specifies the name of the generated Microsoft Excel file. If no name is specified, Microsoft Excel prompts you to enter one with a **Save As** window.

Remarks

Histograms are stored in the ISYSCOLSTAT system table and can also be retrieved with the sa_get_histogram stored procedure. The Histogram utility converts a histogram into a Microsoft Excel chart containing information about the selectivity of predicates. The Histogram utility (dbhist) only works on Windows, and you must have Microsoft Excel 97 or later installed.

Statistics (including histograms) may not be present for a table or materialized view, for example, if statistics were recently dropped. In this case, the Histogram utility returns the message `Histogram contains no data, aborting`. In this case, you must create the statistics, and then run the Histogram utility again. To create statistics for a table or materialized view, execute a CREATE STATISTICS statement. See "CREATE STATISTICS statement" [*SQL Anywhere Server - SQL Reference*].

To determine the selectivity of a predicate over a string column, you should use the ESTIMATE or ESTIMATE_SOURCE functions. Attempting to retrieve a histogram from string columns causes both sa_get_histogram and the Histogram utility to generate an error.

The sheets are named with the column name. Column names are truncated after 24 characters, and all occurrences of \, /, ?, *, [,], and : (which are not allowed in Microsoft Excel) are replaced with underscores (_). Chart names are prefixed with the word chart, followed by the same naming convention above. Duplicate names (arising from character replacement, truncation, or columns named starting with chart) result in a Microsoft Excel error stating that no duplicate names can be used. However, the spreadsheet is still created with those names created with their previous version (Sheet1, Chart1, and so on).

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

You can also retrieve histograms using the sa_get_histogram stored procedure.

Example

The following command (entered all on the same line) generates a Microsoft Excel chart for the column ProductID in the table SalesOrderItems for database *demo.db*, and saves it as *histogram.xls*.

```
dbhist -c "UID=DBA;PWD=sql;DBF=samples-dir\demo.db" -n ProductID -t
SalesOrderItems histogram.xls
```

The following statement generates charts for every column with a histogram in the table SalesOrders, assuming that the sample database is already started. This statement also attempts to connect using UID=DBA and PWD=sql. No output file name is specified, so Microsoft Excel prompts you to enter one.

```
dbhist -t SalesOrders -c "UID=DBA;PWD=sql"
```

For information about *samples-dir*, see "Samples directory" on page 392.

See also

- "sa_get_histogram system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Selectivity estimate sources" [*SQL Anywhere Server - SQL Usage*]
- "Optimizer estimates and column statistics" [*SQL Anywhere Server - SQL Usage*]
- "ESTIMATE function [Miscellaneous]" [*SQL Anywhere Server - SQL Reference*]
- "ESTIMATE_SOURCE function [Miscellaneous]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE STATISTICS statement" [*SQL Anywhere Server - SQL Reference*]

Information utility (dbinfo)

Displays information about the specified database.

Syntax

dbinfo [*options*]

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.

Option	Description
-c *"keyword=value; ..."*	Specifies connection parameters. See "Connection parameters" on page 265. Any valid user ID can run the Information utility, but to obtain page usage statistics you need DBA authority.
-o *filename*	Writes output messages to the named file.
-q	Runs in quiet mode—messages are not displayed.
-u	Displays information about the usage and size of all tables, including system and user-defined tables and materialized views. You can only request page usage statistics if no other users are connected to the database and you have DBA authority. The page usage information is obtained using the sa_table_page_usage system procedure.

Remarks

The dbinfo utility displays information about a database. It reports the name of the database file, the name of any transaction log file or log mirror, the page size, the collation name and label, whether table encryption is enabled, and other information. Optionally, it can also provide table usage statistics and details.

You can use the dbinfo utility to determine the size of a table on disk. To do so, run a command similar to the following:

```
dbinfo -u -c "UID=DBA;PWD=sql;DBF=sample-dir\demo.db"
```

The result shows you how many pages are used to hold the data in each table in your database (Pages), and the percentage used of those pages (%used). For any table, you can then multiply the number of pages by the database page size, and then multiply that by %used to determine the amount of space is being used for your table.

You can also get more information about a database by:

- querying individual database properties using the DB_PROPERTY system function. See "DB_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].
- querying all database properties using the sa_db_properties system procedure. See "sa_db_properties system procedure" [*SQL Anywhere Server - SQL Reference*].
- using the -pd option with the Ping utility (dbping). When you specify -pd, dbping returns the value for each database property that is specified. See "Ping utility (dbping)" on page 826.

Exit codes are 0 (success) or non-zero (failure).

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

Initialization utility (dbinit)

Creates a new database.

Syntax

dbinit [*options*] *new-database-file*

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-a	Causes string comparisons to respect accent differences between letters (for example, e is less than é if the Unicode Collation Algorithm (UCA) is used for either CHAR or NCHAR data types (see -z and -zn). With the exception of Japanese databases created with a UCA collation, by default, accents are ignored (meaning e is equal to é). If all base letters (letters with accents and case removed) are otherwise equal, then accents are compared from left to right. The default accent sensitivity of a UCA collation when creating a Japanese database is *sensitive*. That is, accents are respected. See "Unicode Collation Algorithm (UCA)" on page 420.

Option	Description
-af	Causes string comparisons to respect accent differences between letters (for example, e is less than é) if the UCA is used for either CHAR or NCHAR data types (see -z and -zn). By default, accents are ignored (meaning e is equal to é). If all base letters (letters with accents removed) are otherwise equal, then accents are compared from right to left, consistent with the rules of the French language. For more information, see "Unicode Collation Algorithm (UCA)" on page 420.
-b	Blank pads the database. SQL Anywhere compares all strings as if they are varying length and stored using the VARCHAR domain. This includes string comparisons involving fixed length CHAR or NCHAR columns. In addition, SQL Anywhere never trims or pads values with trailing blanks when the values are stored in the database. By default, SQL Anywhere treats blanks as significant characters. So, the value 'a ' (the character 'a' followed by a blank) is not equivalent to the single-character string 'a'. Inequality comparisons also treat a blank as any other character in the collation. If blank padding is enabled (the dbinit -b option), the semantics of string comparisons more closely follow the ANSI/ISO SQL standard. With blank-padding enabled, SQL Anywhere ignores trailing blanks in any comparison. In the example above, an equality comparison of 'a ' to 'a' in a blank-padded database returns TRUE. With a blank-padded database, fixed-length string values are padded with blanks when they are fetched by an application. The ansi_blanks connection option controls whether the application receives a string truncation warning on such an assignment. See "ansi_blanks option" on page 508.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-c	Considers all values case sensitive in comparisons and string operations. Identifiers in the database are case insensitive, even in case sensitive databases. With the exception of Japanese databases created with a UCA collation, the default behavior is that all comparisons are case *insensitive*. The default case sensitivity of a UCA collation when creating a Japanese database is *sensitive*. Databases used as QAnywhere server stores should be case insensitive. This option is provided for compatibility with the ISO/ANSI SQL standard.

-dba [*DBA-user*][**,***pwd*]	Specifies the DBA user ID and password. If you specify a new name for the DBA user for the database, you can no longer connect to the database as the user DBA. You can also specify a different password for the DBA database user. If you do not specify a password, the default password sql is used. If you do not specify this option, the default user ID DBA with password sql is created.
	
	Either of the following commands creates a database with a DBA user named testuser with the default password sql:
	
	`dbinit -dba testuser mydb.db`
	
	`dbinit -dba testuser, mydb.db`
	
	The following command uses the default user ID DBA with password mypwd:
	
	`dbinit -dba ,mypwd mydb.db`
	
	The following command changes the DBA user to user1 with password mypwd:
	
	`dbinit -dba user1,mypwd mydb.db`
	
	It is recommended that the password be composed of 7-bit ASCII characters as other characters may not work correctly if the server cannot convert from the client's character set to UTF-8.

Option	Description
-dbs *size*[**k** | **m** | **g** | **p**]	Pre-allocates space for the database. Pre-allocating space for the database helps reduce the risk of running out of space on the drive the database is located on. As well, it can help improve performance by increasing the amount of data that can be stored in the database before the database server needs to grow the database, which can be a time-consuming operation. By default, the *size* is in bytes. You can use **k**, **m**, **g**, or **p** to specify units of kilobytes, megabytes, or gigabytes, or pages, respectively.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-ea *algorithm*	Specifies the encryption algorithm used for database or table encryption (-et). Specify `-ea simple` for simple encryption (do not specify -ek or -ep). Simple encryption is equivalent to obfuscation and is intended only to keep data hidden in the event of casual direct access of the database file, to make it more difficult for someone to decipher the data in your database using a disk utility to look at the file. For greater security, specify AES or AES256 for 128-bit or 256-bit strong encryption, respectively. Specify AES_FIPS or AES256_FIPS for 128-bit or 256-bit FIPS-approved strong encryption, respectively. For strong encryption, you must also specify the -ek or -ep option. For more information about strong encryption, see "Strong encryption" on page 1131. To create a database that is not encrypted, specify `-ea none`, or do not include the -ea option (and do not specify -et, -ep, or -ek). If you do not specify the -ea option, the default behavior is as follows: • -ea none, if -ek, -ep, or -et is not specified • -ea AES, if -ek or -ep is specified (with or without -et) • -ea simple, if -et is used without -ek or -ep Algorithm names are case insensitive. On Windows Mobile, the AES_FIPS and AES256_FIPS algorithms are only supported with ARM processors. The following command creates a strongly encrypted database and specifies the encryption key and algorithm. `dbinit -ek "0kZ2o56AK#" -ea AES_FIPS "myencrypteddb.db"` File compression utilities cannot compress encrypted database files as much as unencrypted ones.

Option	Description
	Separately licensed component required ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations. See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].
-ek *key*	Specifies that you want to create a strongly encrypted database by specifying an encryption key directly in the command. The -ek option is used with an AES algorithm, optionally specified using the -ea option. If you specify the -ek option without specifying the -ea option, AES is used by default. When specified with -et, the database is not encrypted. Instead, table encryption is enabled. See "Table encryption" on page 1139. **Caution** For strongly encrypted databases, be sure to store a copy of the key in a safe location. If you lose the encryption key there is no way to access the data, even with the assistance of technical support. The database must be discarded and you must create a new database.
-ep	Specifies that you want to create a strongly encrypted database by inputting the encryption key in a window. This provides an extra measure of security by never allowing the encryption key to be seen in clear text. You must input the encryption key twice to confirm that it was entered correctly. If the keys don't match, the initialization fails. When specified with -et, the database is not encrypted. Instead, table encryption is enabled. For more information, see "Strong encryption" on page 1131.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-et	Enables table encryption using the encryption algorithm (and key) specified for the -ea option. Use this option when you want to create encrypted tables instead of encrypting the entire database. If you specify -et with -ek or -ep, but not -ea, the AES algorithm is used by default. When you specify only -et, simple encryption is used. Enabling table encryption does not mean your tables are encrypted. You must encrypt tables individually, after database creation. See "Encrypting a table" on page 1140. When table encryption is enabled, table pages for the encrypted table, associated index pages, and temporary file pages are encrypted, and the transaction log pages that contain transactions on encrypted tables. The following example creates the database *new.db* with strong encryption enabled for tables using the key abc, and the AES_FIPS encryption algorithm: `dbinit -et -ek abc -ea AES_FIPS new.db`
-i	Excludes jConnect system objects from the database. If you want to use the jConnect JDBC driver to access system catalog information, you need to install jConnect catalog support (it is installed by default). When you specify this option you can still use JDBC, as long as you do not access system information. If you want, you can add jConnect support at a later time using Sybase Central or the ALTER DATABASE statement. For more information, see "Installing jConnect system objects into a database" [*SQL Anywhere Server - Programming*]. If you are creating a database for use on Windows Mobile, see "Using jConnect on Windows Mobile" on page 354.

Option	Description
-k	Does not create the SYSCOLUMNS and SYSINDEXES views. By default, database creation generates the views SYS.SYSCOLUMNS and SYS.SYSINDEXES for compatibility with system tables that were available in Watcom SQL (versions 4 and earlier of this software). These views conflict with the Sybase Adaptive Server Enterprise compatibility views dbo.syscolumns and dbo.sysindexes.
-l	Lists the recommended collation sequences and then stops. No database is created. A list of available collation sequences is automatically presented in the Sybase Central **Create Database Wizard**.
-le	Lists the available character set encodings and then stops. No database is created. Each character set encoding is identified by one or more labels. These are strings that can be used to identify the encoding. Each line of text that appears lists the encoding label and alternate labels by which the encoding can be identified. These labels fall into one of several common categories: SA (the SQL Anywhere label), IANA (Internet Assigned Numbers Authority), MIME (Multipurpose Internet Mail Extensions), ICU (International Components for Unicode), JAVA, or ASE (Adaptive Server Enterprise). If you want to view a list of character set encodings that includes the alternate labels, specify the -le+ option. When the Initialization utility reports the character set encoding, it always reports the SQL Anywhere version of the label. For example, the following command reports the CHAR character set encoding windows-1250: `dbinit -ze cp1250 -z uca test.db`
-m *filename*	Creates a transaction log mirror. A transaction log mirror is an identical copy of a transaction log, usually maintained on a separate device, for greater protection of your data. By default, SQL Anywhere does not use a transaction log mirror.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-n	Creates a database without a transaction log. Creating a database without a transaction log saves disk space, but can result in poorer performance because each commit causes a checkpoint. Also, if your database becomes corrupted and you are not running with a transaction log, data is not recoverable. The transaction log is required for data replication and provides extra security for database information during a media or system failure.
-o *filename*	Writes output messages to the named file.
-p *page-size* [**k**]	Specifies the page size for the database. The page size for a database can be (in bytes) 2048, 4096, 8192, 16384, or 32768, with 4096 being the default. Use **k** to specify units of kilobytes (for example, -p 4k). Large databases can benefit from a larger page size. For example, the number of I/O operations required to scan a table is generally lower, as a whole page is read in at a time. However, there are additional memory requirements for large page sizes. It is strongly recommended that you do performance testing (and testing in general) when choosing a page size. Then choose the smallest page size that gives satisfactory results. For most applications, 16 KB or 32 KB page sizes are not recommended. You should not use page sizes of 16 KB or 32 KB in production systems unless you can be sure that a large database server cache is always available, and only after you have investigated the trade offs of memory and disk space with its performance characteristics. If a large number of databases are going to be started on the same server, pick a reasonable page size. For more information, see: • "Use an appropriate page size" [*SQL Anywhere Server - SQL Usage*] • "Table and page sizes" [*SQL Anywhere Server - SQL Usage*]
-q	Runs in quiet mode—messages are not displayed.

Option	Description
-s[**-**]	Adds global checksums (a checksum is added to each database page). By default, this option is on. Checksums are used to determine whether a database page has been modified on disk. When you create a database with global checksums enabled, a checksum is calculated for each page just before it is written to disk. The next time the page is read from disk, the page's checksum is recalculated and compared to the checksum stored on the page. If the checksums are different, then the page has been modified or corrupted on disk, and an error occurs. Critical database pages are always checksummed by the database server, regardless of the value of the -s option. Checksums are automatically enabled for databases running on Windows Mobile and storage devices such as removable drives to help provide early detection if the database becomes corrupt. If a database is created with global checksums disabled, you can still add checksums to pages when they are written by using the -wc option or the START DATABASE statement. See "Using checksums to detect corruption" on page 928.
-t *transaction-log-name*	Specifies the name of the transaction log file. The transaction log is a file where the database server logs all changes, made by all users, no matter what application is being used. The transaction log plays a key role in backup and recovery, and in data replication. If the file name has no path, it is placed in the same directory as the database file. If you run dbinit without specifying -t or -n, a transaction log is created with the same file name as the database file, but with extension *.log*. For more information, see "The transaction log" on page 21.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-z *coll* [*collation-tailoring-string*]	Specifies the collation sequence for the database. The collation sequence is used for sorting and comparing character data types (CHAR, VARCHAR, and LONG VARCHAR). The collation provides character comparison and ordering information for the encoding (character set) being used. It is important to choose your collation carefully. It cannot be changed after the database has been created without unloading and reloading the database. If the collation is not specified, SQL Anywhere chooses a collation based on the operating system language and character set. See: • "Choosing collations" on page 426 • "Recommended character sets and collations" on page 436 • "Supported and alternate collations" on page 423 Optionally, you can specify collation tailoring options (*collation-tailoring-string*) for additional control over the sorting and comparing of characters. These options take the form of keyword=value pairs, assembled in parentheses, following the collation name. For example: `dbinit -c -z uca(locale=es;case=LowerFirst) spanish2.db` See "Collation tailoring options" on page 427. Case and accent settings specified in the *collation-tailoring-string* override case and accent options for dbinit (-c, -a, and -af), if you specify both. **Note** Databases initialized with collation tailoring options cannot be started by a pre-10.0.1 database server.

Option	Description
-ze *encoding*	Specifies the encoding for the collation. Most collations specified by -z dictate both the encoding (character set) and ordering. For those collations, -ze should not be specified. If the collation specified by -z is Unicode Collation Algorithm (UCA), then -ze can specify UTF-8 or any single-byte encoding for CHAR data types. By default, SQL Anywhere uses UTF-8. Use -ze to specify a locale-specific encoding and get the benefits of the UCA for comparison and ordering.
-zn *coll* [*collation-tailoring-string*]	Specifies the collation sequence used for sorting and comparing of national character data types (NCHAR, NVARCHAR, and LONG NVARCHAR). The collation provides character ordering information for the UTF-8 encoding (character set) being used. Values are UCA (the default), or UTF8BIN which provides a binary ordering of all characters whose encoding is greater than 0x7E. If the dbicu12 and dbicudt12 DLLs are not installed, then the default NCHAR collation is UTF8BIN. For more information, see "Choosing collations" on page 426. Optionally, you can specify collation tailoring options (*collation-tailoring-string*) for additional control over the sorting and comparing of characters. These options take the form of keyword=value pairs, assembled in parentheses, following the collation name. For example: `dbinit -c -zn UCA(case=LowerFirst) sens.db` See "Collation tailoring options" on page 427. Case and accent settings specified in the *collation-tailoring-string* override case and accent options for dbinit (-c, -a, and -af), if you specify both. **Note** Databases initialized with collation tailoring options cannot be started by a pre-10.0.1 database server.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

Several database attributes are specified at initialization and cannot be changed later except by unloading, reinitializing, and rebuilding the entire database. These database attributes include:

- Case sensitivity or insensitivity
- Accent sensitivity or insensitivity
- Punctuation sensitivity
- Treatment of trailing blanks in comparisons
- Page size
- Character set encoding or collation sequence
- Database encryption
- Table encryption

For example, the database *test.db* can be created with 8192 byte pages as follows:

```
dbinit -p 8192 test.db
```

You cannot name a database **utility_db**. This name is reserved for the utility database. See "Using the utility database" on page 28.

When specifying collation tailoring options in the initialization command, you cannot specify quaternary for the punctuation sensitivity if the database is case or accent insensitive.

In addition, the choice of whether to use a transaction log and a transaction log mirror is made at initialization. This choice can be changed later using the Transaction Log utility or the ALTER DATABASE statement.

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

You can also create a database in the following ways:

- From Sybase Central, using the **Create Database Wizard**. See "Create a database (Sybase Central)" on page 9.
- From Interactive SQL, using the CREATE DATABASE statement. See "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

Note
When you are deploying applications, the personal database server (dbeng12) is required for creating databases using the dbinit utility. It is also required if you are creating databases from Sybase Central on the local computer when no other database servers are running.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

Example

The following command creates a case sensitive database, *spanish.db*, which uses the 1262spa collation for non-NCHAR data. For NCHAR data, the UCA collation is specified, with locale es, and sorting by lowercase first.

```
dbinit -c -z 1252spa -zn uca(locale=es;case=LowerFirst) spanish.db
```

Interactive SQL utility (dbisql)

Executes SQL commands and runs command files against a database.

Syntax

dbisql -c "*connection-string*" [*options*] [*dbisql-command* | *command-file*]

dbisql-command: A SQL statement or a series of sql statements separated by a command-delimiter.

Option	Description
@*data*	Reads in options from the specified environment variable or configuration file. If both the environment variable and configuration file exist with the same name, the environment variable is used. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-c *"keyword=value; ..."*	Specifies connection parameters. If Interactive SQL cannot connect, you are presented with a window where you can enter the connection parameters. If you do not specify both a user ID and a password, the default UID of **DBA** and PWD of **sql** are assumed. See "Connection parameters" on page 265.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-d *delimiter*	Specifies a command delimiter. Quotation marks around the delimiter are optional, but are required when the command shell itself interprets the delimiter in some special way. This option overrides the setting of the command_delimiter option. See "command_delimiter option [Interactive SQL]" on page 742.
-d1	Echoes all statements explicitly executed by the user to the command window (STDOUT). This can provide useful feedback for debugging SQL scripts, or when Interactive SQL is processing a long SQL script. (The final character is a number 1, not a lowercase L). This option is only available when you run Interactive SQL as a command line program.
-datasource *DSN-name*	Specifies an ODBC data source to connect to.
-f *filename*	Opens (but does not run) in the SQL Statements pane the file called *filename*. If the -f option is given, the -c option is ignored; that is, no connection is made to the database. The file name can be enclosed in quotation marks, and *must* be enclosed in quotation marks if the file name contains a space. If the file does not exist, or if it is really a directory instead of a file, Interactive SQL prints an error message and then quits. If the file name does not include a full drive and path specification, it is assumed to be relative to the current directory. This option is only supported when Interactive SQL is run as a windowed application.
-host *hostname*	Specifies the *hostname* or IP address of the computer on which the database server is running. You can use the name localhost to represent the current computer.

Option	Description
-nogui	Runs Interactive SQL as a console application, with no windowed user interface. This is useful for batch operations. If you specify either *dbisql-command* or *command-file*, then -nogui is assumed. In this mode, Interactive SQL sets the program exit code to indicate success or failure. On Windows operating systems, the environment variable ERRORLEVEL is set to the program exit code. See "Software component exit codes" [*SQL Anywhere Server - Programming*].
-onerror { **continue** | **exit** }	Controls what happens if an error is encountered while reading statements from a command file. It is useful when using Interactive SQL in batch operations. This option overrides the on_error setting. See "on_error option [Interactive SQL]" on page 752. Define one of the following supported *behavior* values: • **Continue** The error is ignored and Interactive SQL continues executing statements. • **Exit** Interactive SQL terminates.
-port *port-number*	Specifies the port number on which the SQL Anywhere database server is running. The default port number for SQL Anywhere is 2638.
-q	Suppresses output messages. Sets the utility to run in quiet mode. This is useful only if you start Interactive SQL with a command or command file. Specifying this option does not suppress error messages, but it does suppress the following: • warnings and other non-fatal messages • the printing of result sets

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-ul	Specifies that UltraLite databases are the default. Interactive SQL customizes the options available to you depending on the type of database you are connected to. By default, Interactive SQL assumes that you are connecting to SQL Anywhere databases. When you specify the -ul option, the default changes to UltraLite databases. Regardless of the type of database set as the default, you can connect to either SQL Anywhere or UltraLite databases by choosing the database type from the **Change Database Type** dropdown list on the **Connect** window. For more information about connecting to UltraLite databases from Interactive SQL, see "Interactive SQL for UltraLite utility (dbisql)" [*UltraLite - Database Management and Reference*].
-version	Displays the version number of Interactive SQL. You can also view the version number from within Interactive SQL; from the **Help** menu, choose **About Interactive SQL**.
-x	Scans commands but does not execute them. This is useful for checking long command files for syntax errors. For detailed descriptions of SQL statements and Interactive SQL commands, see "SQL language elements" [*SQL Anywhere Server - SQL Reference*].
dbisql-command | *command-file*	Execute the SQL statement or execute the specified *command-file*. If you do not specify a *SQL-statement* or *command-file*, Interactive SQL enters interactive mode, where you can type a command into a command window.

Remarks

Interactive SQL allows you to browse the database, execute SQL commands, and run command files. It also provides feedback about:

- the number of rows affected
- the time required for each command
- the execution plan of queries
- any error messages

You can connect to both SQL Anywhere and UltraLite databases. For information about connecting to UltraLite databases, see "Interactive SQL for UltraLite utility (dbisql)" [*UltraLite - Database Management and Reference*].

Interactive SQL is supported on Windows, Solaris, Linux, and Mac OS X. See http://www.sybase.com/detail?id=1061806.

For Windows, there are two executables:

1. Batch scripts should call *dbisql* or *dbisql.com*, not *dbisql.exe*. The *dbisql.com* executable is linked as a console application.

2. The *dbisql.exe* executable is linked as a windowed application and does not block the command shell from which it was started. If *dbisql.exe* is run from a batch file, you won't see any output sent to the standard output or standard error files.

The default encoding for Interactive SQL can also be temporarily set using the default_isql_encoding option. See "default_isql_encoding option [Interactive SQL]" on page 743.

You can specify the encoding to use when reading or writing files using the ENCODING clause of the INPUT, OUTPUT, or READ statement. See:

- "INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "OUTPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "READ statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

Exit codes are 0 (success) or non-zero (failure). Non-zero exit codes are set only when you run Interactive SQL in batch mode (with a command line that contains a SQL statement or the name of a script file). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

When executing a *reload.sql* file with Interactive SQL, you must specify the encryption key as a parameter. If you do not provide the key in the READ statement, Interactive SQL prompts for the key.

You can start Interactive SQL in the following ways:

- From Sybase Central, choosing **File** » **Open Interactive SQL**.
- From the **Start** menu by choosing **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Interactive SQL**.
- Using the dbisql command from a command prompt.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Interactive SQL SQL statements" on page 733
- "Using Interactive SQL" on page 697

Example

The following command runs the command file *mycom.sql* against the current default server, using the user ID DBA and the password sql. If there is an error in the command file, the process shuts down.

```
dbisql -c "UID=DBA;PWD=sql" -onerror exit mycom.sql
```

The following command adds a user to the current default database:

```
dbisql -c "UID=DBA;PWD=sql" CREATE USER joe IDENTIFIED BY passwd
```

Key Pair Generator utility (createkey)

Creates RSA and ECC key pairs for use with MobiLink end-to-end encryption.

Syntax

createkey

Remarks

To create ECC objects, you must order a separate license. See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

When you run createkey, you are prompted for the following information:

- **Choose encryption type** This prompt only appears if you have purchased a license for ECC encryption. Choose RSA or ECC.
- **Enter RSA key length (512-16384)** This prompt only appears if you chose RSA encryption. You can choose a length between 512 bits and 16384 bits.
- **Enter ECC curve** This prompt only appears if you have purchased a license for ECC encryption and you chose the ECC encryption type. You are prompted to choose from a list of ECC curves. The default is sect163k1.
- **Enter file path to save public key** Specify a file name and location for the generated PEM-encoded public key. This file is specified on the MobiLink client by the e2ee_public_key protocol option. See "e2ee_public_key" [*MobiLink - Client Administration*].
- **Enter file path to save private key** Specify a file name and location for the generated PEM-encoded private key. This file is specified on the MobiLink server via the e2ee_private_key protocol option. See "-x mlsrv12 option" [*MobiLink - Server Administration*].
- **Enter password to protect private key** Optionally, supply a password with which to encrypt the private key. The private key is not encrypted if you do not supply a password. This password is

specified on the MobiLink server via the e2ee_private_key_password protocol option. See "-x mlsrv12 option" [*MobiLink - Server Administration*].

See also

- "End-to-end encryption" on page 1158
- "e2ee_type" [*MobiLink - Client Administration*] (MobiLink client network protocol option)

Example

The following example creates an RSA key pair:

```
>createkey
SQL Anywhere Key Pair Generator Version 12.0.0.1304
Choose encryption type ((R)SA or (E)CC): r
Enter RSA key length (512-16384): 2048
Generating key pair...
Enter file path to save public key: rsapublic.pem
Enter file path to save private key: rsaprivate.pem
Enter password to protect private key: pwd
```

Language Selection utility (dblang)

Reports and changes the registry settings that control the languages used by SQL Anywhere, Sybase Central, and Interactive SQL.

Syntax

dblang [*options*] *language-code*

Option	Description
-m	Writes the language code to the registry under HKEY_LOCAL_MACHINE.
-q	Runs in quiet mode—messages are not displayed.
-u	Writes the language code to the registry under HKEY_CURRENT_USER. This is the default location.

Language code	Language
EN	English
DE	German
ES	Spanish
FR	French
IT	Italian

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Language code	Language
JA	Japanese
KO	Korean
LT	Lithuanian
PL	Polish
PT	Portuguese
RU	Russian
TW	Traditional Chinese
UK	Ukrainian
ZH	Simplified Chinese

Remarks

If you do not specify -m or -u, then the language code is written to the registry under HKEY_CURRENT_USER. You can specify both -m and -u to write the language code to both locations.

Running the dblang utility without a language code reports the current settings. These settings are as follows:

- **SQL Anywhere** This setting controls which language resource library is used to deliver informational and error messages from the SQL Anywhere database server. The language resource library is a DLL with a name of the form *dblgXX12.dll*, where *XX* is a two-letter language code.

  Ensure that you have the appropriate language resource library on your computer when you change the settings.

- **Sybase Central and Interactive SQL** This setting controls the resources used to display user interface elements for Sybase Central and Interactive SQL.

  For information about how language changes affect the fast launcher, see "Using the fast launcher option" on page 757.

For information about how language changes affect the fast launcher, see "Using the fast launcher option" on page 757.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

This utility does *not* accept the ***@data*** parameter to read in options from a configuration file.

See also

- "SALANG environment variable" on page 384
- "Language (LANG) connection parameter" on page 295
- "Changing administration tool language settings" on page 758

Example

The following command displays a window containing the current settings:

```
dblang
```

The following command changes the settings to German, and displays a window containing the previous and new settings:

```
dblang de
```

Log Translation utility (dbtran)

Translates a transaction log into a SQL command file.

Syntax

Running against a database server:

dbtran [*options*] **-c** { *connection-string* } **-n** *SQL-file*

Running against a transaction log:

dbtran [*options*] [*transaction-log*] [*SQL-file*]

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-a	Controls whether uncommitted transactions appear in the transaction log. The transaction log contains changes made before the most recent COMMIT by any transaction. Changes made after the most recent commit are not present in the transaction log. If -a is not specified, only committed transactions appear in the output file. If -a is specified, any uncommitted transactions found in the transaction log are output followed by a ROLLBACK statement.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-c *"keyword=value; ..."*	Specifies the connection string when running the utility against a database server. See "Connection parameters" on page 265.
-d	Specifies that transactions are written in order from earliest to latest. This feature is provided primarily for use when auditing database activity: the output of dbtran should not be applied against a database.
-ek *key*	Specifies the encryption key for strongly encrypted databases. If you have a strongly encrypted database, you must provide the encryption key to use the database or transaction log. For strongly encrypted databases, you must specify either -ek or -ep, but not both. The command fails if you do not specify the correct encryption key. If you are running dbtran against a database server using the -c option, specify the key using a connection parameter instead of using the -ek option. For example, the following command gets the transaction log information about database *myenc.db* from the database server **sample**, and saves its output in *log.sql*. `dbtran -n log.sql -c "Server=sample;DBF=myenc.db; UID=DBA;PWD=sql;DBKEY=mykey"`
-ep	Prompts for the encryption key. This option causes a window to appear, in which you enter the encryption key. It provides an extra measure of security by never allowing the encryption key to be seen in clear text. For strongly encrypted databases, you must specify either -ek or -ep, but not both. The command fails if you do not specify the correct encryption key. If you are running dbtran against a database server using the -c option, specify the key using a connection parameter, instead of using the -ep option. For example, the following command gets the transaction log information about database *myenc.db* from the database server **sample**, and saves its output in *log.sql*. `dbtran -n log.sql -c "Server=sample;DBF=myenc.db; UID=DBA;PWD=sql;DBKEY=mykey"`
-f	Outputs only transactions that were completed since the last checkpoint.
-g	Adds auditing information to the transaction log if the auditing database option is turned on. You can include this information as comments in the output file using this option. See "auditing option" on page 514. The -g option implies the -a, -d, and -t options.
-ir *offset1,offset2*	Outputs a portion of the transaction log between two specified offsets.

Option	Description
-is *source,...*	Outputs operations on rows that have been modified by operations from one or more of the following sources, specified as a comma-separated list: • **All** All rows. This is the default setting. • **SQLRemote** Include only rows that were modified using SQL Remote. You can also use the short form SR. • **Local** Include only rows that are not replicated.
-it *owner.table,...*	Outputs those operations on the specified, comma-separated list of tables. Each table should be specified as *owner.table*.
-j *date/time*	Translates only transactions from the most recent checkpoint before the given date and/or time. The user-provided argument can be a date, time, or date and time, enclosed in quotes. If the time is omitted, the time is assumed to be the beginning of the day. If the date is omitted, the current day is assumed. The following is an acceptable format for the date and time: *"YYYY/MMM/DD HH:NN"*.
-k	Prevents partial *.sql* files from being erased if an error is detected. If an error is detected while dbtran is running, the *.sql* file generated until that point is normally erased to ensure that a partial file is not used by accident. Specifying this option may be useful if you are attempting to salvage transactions from a damaged transaction log.
-m	Specifies a directory that contains transaction logs. This option must be used in conjunction with the -n option.
-n *filename*	Specifies the output file that holds the SQL statements when you run the dbtran utility against a database server.
-o *filename*	Writes output messages to the named file.
-q	Runs in quiet mode—messages are not displayed.
-r	Removes any transactions that were not committed. This is the default behavior.
-s	Controls how UPDATE statements are generated. If the option is not used, and there is no primary key or unique index on a table, the Log Translation utility generates UPDATE statements with a non-standard FIRST keyword if there are duplicate rows. If the option is used, the FIRST keyword is omitted for compatibility with the SQL standard.
-sr	Places generated comments in the output file describing how SQL Remote distributes operations to remote sites.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-t	Controls whether triggers are included in the command file. By default, actions performed by triggers are not included in the command file. If the matching trigger is in the database, when the command file is run against the database the trigger performs the actions automatically. Trigger actions should be included if the matching trigger does not exist in the database against which the command file is to run.
-u *userid,...*	Limits the output from the transaction log to include only specified users.
-x *userid,...*	Limits the output from the transaction log to exclude specified users.
-y	Replaces existing command files without prompting you for confirmation. If you specify -q, you must also specify -y or the operation fails.
-z	Includes transactions that were generated by triggers only as comments in the output file.
transaction-log	Specifies the log file to be translated. Cannot be used together with -c or -m options.
SQL-file	Names the output file containing the translated information. For use with *transaction-log* only.

Remarks

The dbtran utility takes the information in a transaction log and places it as a set of SQL statements and comments into an output file. The utility can be run in the following ways:

- **Against a database server** When dbtran is run against a database server, the utility is a standard client application. It connects to the database server using the connection string specified following the -c option, and places output in a file specified with the -n option. DBA authority is required to run in this way.

  The following command translates log information from the server **demo12** and places the output in a file named *demo.sql*.

  ```
  dbtran -c "Server=demo12;DBN=demo;UID=DBA;PWD=sql" -n demo.sql
  ```

- **Against a transaction log file** When dbtran is run against a transaction log, the utility acts directly against a transaction log file. You should protect your transaction log file from general access if you want to prevent users from having the capability of running this statement.

  ```
  dbtran demo.log demo.sql
  ```

When the dbtran utility runs, it displays the earliest log offset in the transaction log. This can be an effective method for determining the order in which multiple log files were generated.

If -c is used, dbtran attempts to translate the online transaction log file, and all the offline transaction log files in the same directory as the online transaction log file. If the directory contains transaction log files for more than one database, dbtran may give an error. To avoid this problem, ensure that each directory contains transaction log files for only one database.

A transaction can span multiple transaction logs. If transaction log files contain transactions that span logs, translating a single transaction log file (for example `dbtran demo.log`) can cause the spanning transactions to be lost. In order for dbtran to generate complete transactions, use the -c or -m options with the transaction log files in the directory. See "Recovering a database with multiple transaction logs" on page 907.

You can access the Log Translation utility in the following ways:

- From Sybase Central, using the **Translate Log File Wizard**.
- At a command prompt, using the dbtran command. This is useful for incorporation into batch or command files.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

Performance Statistics utility (dbstats) (Unix)

Returns performance statistics for SQL Anywhere database servers on Unix.

Syntax

dbstats [*options*] [*interval*] [*iterations*]

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-d *statistic-definition-string*	Specifies the statistic name and the scope (database server, database, or connection) at which the statistic is monitored. For a complete list of statistics that can be monitored, see "Performance Monitor statistics" [*SQL Anywhere Server - SQL Usage*].
-e *server-definition-string*	Specifies a comma-separated list of database server names to monitor. If this option is not specified, the dbstats utility reports statistics only for the default database server.
-l	Lists the statistics that are available to be monitored.
-o	Writes output messages to the named file.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-v *verbosity*	Reports messages at the specified level. The supported values are **error**, **warning**, and **info**. The default level is info.
interval	Specifies how often, in seconds, statistics are reported. If you only specify *interval*, then the utility returns the value of statistics at the specified interval until you stop the reporting of statistics.
iterations	Specifies the total number of times statistics are reported. You must specify an *interval* value if you specify a value for *iterations*.

Remarks

The dbstats utility uses a shared memory connection to monitor SQL Anywhere servers running on the same computer. This utility cannot be used to monitor remote database servers.

A *statistic-definition-string* takes the following form:

[[*statistic-name*][,[*statistic-name*],[...]]] [:[*database-servers*]/[*databases*]/[*connections*]]

The values *database-servers*, *databases*, and *connections* are comma-separated lists of database server names, database names, and connection names to monitor. In addition to specifying a comma-separated list of names, you can specify an asterisk (*), which indicates that all database servers, databases, and connections should be monitored. If you do not specify a value for one of the supported scopes, then that scope is not monitored.

If no options are specified for the dbstats utility, the utility automatically monitors the statistics ActiveReq, ConnCount, CurrentCacheSize, DiskRead, DiskWrite, MainHeapPages, and UnschReq, for all scopes in which they are valid.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

See also

- "Performance Monitor statistics" [*SQL Anywhere Server - SQL Usage*]
- "Connection, database, and database server properties" on page 619

Example

Query only the ActiveReq statistic for all databases and connections on the default database server and then exit:

```
dbstats -c ActiveReq
```

Query only the ActiveReq statistic for all databases and connections on the default database server, once per second forever:

```
dbstats -c ActiveReq 1
```

Query only the ActiveReq statistic for all databases and connections on the default database server five times, once per second, then exit:

```
dbstats -c ActiveReq 1 5
```

Query the ActiveReq and ConnCount statistics for all databases and connections and then exit:

```
dbstats -c "[ ActiveReq, ConnCount ]"
```

Query the ActiveReq and ConnCount statistics at the database server scope only and then exit:

```
dbstats -c "[ActiveReq, ConnCount]:*"
```

Query the ActiveReq and ConnCount statistics at the database scope only for databases named db1 and db2 running on the default server and then exit:

```
dbstats -c "[ActiveReq, ConnCount]:/db1,db2/"
```

Ping utility (dbping)

Locates database servers and tests connections to databases.

Syntax

dbping [*options*]

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-c *"keyword=value; ..."*	Specifies connection parameters that control the behavior of dbping. If connection parameters are not specified, connection parameters from the SQLCONNECT environment variable are used, if set. See "Connection parameters" on page 265. If you use the following command to start dbping and there is a database server named demo12 already running, dbping attempts to connect to a database named demo. If no such database is running on that database server, the database server attempts to load *demo.db*. If no server called demo12 is found, dbping attempts to start one automatically. `dbping -d -c "UID=DBA;PWD=sql;Server=demo12;DBN=demo;DBF=samples-dir\demo.db"` For information about *samples-dir*, see "Samples directory" on page 392.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description

-d	Pings the database, not just the server.
	
	If you supply the -d option, then dbping reports success only if it connects to the server and also connects to a database. If you do not supply the -d option, then dbping reports success if it finds the server specified by the -c option.
	
	For example, if you have a database server named blair running the sample database, the following succeeds:
	
	`dbping -c "Server=blair;DBN=demo"`
	
	The following command fails, with the message `Ping database failed -- specified database not found`:
	
	`dbping -c "Server=blair;DBN=demo" -d`

-en	Specifies that you want dbping to exit with a failed return code when NULL is returned for any of the properties specified. By default, dbping prints NULL when the value for a property specified by -pc, -pd, or -ps is unknown, and exits with a success return code. This option can only be used in conjunction with -pc, -pd, and -ps.
-l *library*	Specifies the library to use (without its file extension). This option avoids the use of the ODBC driver manager, and so is useful on Unix operating systems. For example, the following command loads the ODBC driver directly: `dbping -m -c "DSN=SQL Anywhere 12 Demo" -l dbodbc12` On Unix, if you want to use a threaded connection library, you must use the threaded version of the Ping utility, dbping_r.
-m	Establishes a connection using ODBC. By default, the utility connects using the embedded SQL interface.
-o *filename*	Writes output messages to the named file.
-pc *property,...*	Displays the specified connection properties. Supply the properties in a comma-separated list. You must specify enough connection information to establish a database connection if you use this option. See “Connection properties” on page 619. For example, the following command displays the fire_triggers option setting, which is available as a connection property. `dbping -c ... -pc fire_triggers`

Option	Description

-pd *property*[**@***db-name*],...	Displays the specified database properties. Supply the properties in a comma-separated list. See "Database properties" on page 659.
	
	For example, the following command displays the page size in use by the database:
	
	`dbping -c ... -pd PageSize`
	
	Optionally, you can specify the name of a database running on the database server you want to obtain the value from. For each property listed, if the database name is not specified by appending **@***db-name* to the property, then the database name used for the previous property is used.
	
	The following command displays the page size and collation of the database mydb:
	
	`dbping -c ... -pd PageSize@mydb,Collation`

-ps *property*,...	Displays the specified database server properties. Supply the properties in a comma-separated list. You must specify enough connection information to establish a database connection if you use this option. See "Database server properties" on page 644. For example, the following command displays the number of licensed seats or processors for the database server: `dbping -c ... -ps LicenseCount`
-q	Runs in quiet mode—messages are not displayed.
-s	Returns information about the performance of the network between the computer running dbping and the computer running the database server. Approximate connection speed, latency, and throughput are displayed. The -c option is usually required to specify the connection parameters to connect to a database on the server. You can only use dbping -s for embedded SQL connections. This option is ignored if -m or -l is also specified. By default, dbping -s loops through the requests for at least one second for each statistic it measures. A maximum of 200 connect and disconnect iterations are performed, regardless of the time they take, to avoid consuming too many resources. On slower networks, it can take several seconds to perform the minimum number of iterations for each statistic. The performance statistics are approximate, and are more accurate when both the client and server computers are fairly idle. See "Testing embedded SQL connection performance" on page 146.
-st *time*	Returns the same information as the -s option, except that the -st option specifies the length of time, in seconds, that dbping loops through the requests for each statistic it measures. This option allows more accurate timing information to be obtained that -s. See "Testing embedded SQL connection performance" on page 146.
-z	Displays the network communication protocols used to attempt connection, and other diagnostic messages. This option is available only when an embedded SQL connection is being attempted. That is, it cannot be combined with -m or -l.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

The dbping utility is a tool to help debug connection problems. It takes a full or partial connection string and returns a message indicating whether the attempt to locate a server or database, or to connect, was successful.

The utility can be used for embedded SQL or ODBC connections. It cannot be used for jConnect (TDS) connections.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

Script Execution utility (dbrunsql)

Allows you to execute SQL commands and run command files against databases running on Windows Mobile.

Syntax

dbrunsql [*options*] [*SQL-script-file* | *SQL-command*]

Option	Description
-c *"keyword=value; ..."*	Specifies connection parameters. See "Connection parameters" on page 265.
-d	Writes data exported from result sets to the output file. If you do not specify -d, then all dbrunsql output is written to the output file.
-e [**c** | **p** | **s**]	Controls the behavior when an error is encountered while executing statements. By default, dbrunsql prompts the user when an error occurs. • **c** Ignore the error and continue executing statements. • **p** Prompt the user to see if the user wants to continue. • **s** Stop executing statements.
-f [**f** | **a**]	Specifies the data format dbrunsql uses to export result sets. You can specify one of the following values: • **a** Use ASCII format when exporting data. • **f** Use FIXED format when exporting data. This is the default format.

Option	Description
-g [+ | -]	Prevents the GUI from appearing. By default, the dbrunsql GUI appears.
-o *filename*	Writes output messages to the named file.
-q	Suppresses output messages. This is useful only if you start Interactive SQL with a command or command file. Specifying this option does not suppress error messages, but it does suppress the following: • warnings and other non-fatal messages • the printing of result sets
-qc	Closes the dbrunsql window once the command or script file has been executed.
-s *number*	Specifies the maximum number of bytes fetched per column when you are exporting result sets using the FIXED format. The default value is 255.
-v	Includes all lines of each SQL statement in the dbrunsql output. Otherwise, when you execute a script file, the number of the line that is currently being executed appears.

Remarks

The dbrunsql utility allows you to execute SQL commands or run command files against a database. The SQL Anywhere Script Execution utility (dbrunsql) is only supported on Windows Mobile.

Server Enumeration utility (dblocate)

Locates database servers on the TCP/IP network.

Syntax

dblocate [*options*] [*host*]

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-d	Lists the server name and address, for each server found, followed by a comma-separated list of databases running on that server. If the list exceeds 160 characters, it is truncated and ends with an ellipsis (...). Databases that are running on SQL Anywhere 9.0.2 and earlier database servers or that were started with the -dh database option are not listed. See "-dh dbeng12/dbsrv12 database option" on page 255.
-dn *database-name*	Lists the server name and address, for servers running a database with the specified name. If the list exceeds 160 characters, it is truncated and ends with an ellipsis (...). Databases that are running on SQL Anywhere 9.0.2 and earlier database servers or that were started with the -dh database option are not listed. See "-dh dbeng12/dbsrv12 database option" on page 255.
-dv	Displays the server name and address, for each server found, listing each database running on that server on a separate line. The list is not truncated, so this option can be used to reveal lists that are truncated when the -d option is used. Databases that are running on SQL Anywhere 9.0.2 and earlier database servers or that were started with the -dh database option are not listed. See "-dh dbeng12/dbsrv12 database option" on page 255.
-n	Lists IP addresses in the output, rather than computer names. This may improve performance since looking up computer names may be slow.
-o *filename*	Writes output messages to the named file.

Option	Description
-p *port-number*	Displays the server name and address only for servers using the specified TCP/IP port number. The TCP/IP port number must be between 1 and 65535.
-q	Runs in quiet mode—messages are not displayed.
-s *name*	Displays the server name and address only for servers with the specified server name. If this option is used, the -ss option should not be used (if both options are used, it is likely that no matching servers will be found).
-ss *substr*	Displays the server name and address only for servers that contain the specified substring anywhere in the server name. If this option is used, the -s option should not be used (if both options are used, it is likely that no matching servers will be found).
-v	Displays the full server name. By default, dblocate truncates database server names that are longer than 40 bytes. Version 9.0.2 and earlier clients, including dblocate, cannot connect to version 10.0.0 and later database servers with names longer than 40 bytes.
host	Lists only database servers running on the computer with the specified IP address or host name. For example, the following command looks for servers on the computer jfrancis: `dblocate jfrancis` The hostname or IP address can be of any format, regardless of whether -n is specified. For example, consider a server is running on myhost.mycompany.com, which has an IP address of 1.2.3.4. To list only servers running on this computer from any computer with the mycompany.com domain, any of `dblocate myhost`, `dblocate myhost.mycompany.com`, or `dblocate 1.2.3.4` can be used.

Remarks

The Server Enumeration utility (dblocate) locates any SQL Anywhere database servers running over TCP/IP on the immediate network, and prints a list of the database servers and their addresses. This list includes alternate server names. See "-sn dbsrv12 database option" on page 260.

Depending on your network, it may take several seconds for dblocate to prints its results.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Note
If a database server is using a TCP/IP port other than 2638 on Mac OS X, dblocate will not find it, even if the -p option is used to specify the TCP/IP port. See "ServerPort (PORT) protocol option" on page 335.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

The database server can register itself with an LDAP server, which keeps track of all servers in an enterprise. This allows both clients and dblocate to find them, regardless of whether they are on a WAN or LAN, through firewalls, and without specifying an IP address. LDAP is only used with TCP/IP, and only on network servers. See "Connecting using an LDAP server" on page 81.

If the same database server name is found more than once, dblocate displays the IP address of each host, even if the -n option is not specified. The same server name could be found when a server is running on a computer with multiple IP addresses (for example, if the computer has multiple network cards), or if a network server is running on a remote computer and a personal server with the same name is running on the local computer.

Server Licensing utility (dblic)

Applies your software license to your SQL Anywhere database server or MobiLink server.

Syntax

dblic [*options*] *license-file* **"***user-name***" "***company-name***"**

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-k *registration-key*	Specifies a SQL Anywhere license key. You can use this option to change your database server's edition. For example, if you have the developer edition and want to change to the workgroup edition, you can specify the workgroup edition registration key that you receive from iAnywhere Solutions.

Option	Description
-l *type*	Specifies the license type that matches the licensing model described in your software license agreement. The following license types are supported: • **Perseat** A perseat license restricts the number of client connections to the database server. With perseat licensing, the network database server uses all CPUs available on the computer unless the network database server is limited by the -gt option or by the edition you are running. The personal server is limited to one CPU. • **Processor** A processor license restricts the number of separate physical processors that can be used by the database server. The number of CPUs that can be used by the database server may be further limited by the -gt option or by the SQL Anywhere edition you are running. The personal database server is limited to one CPU. For this license type, the database server treats each physical processor as a CPU and does not treat a dual core or hyperthreaded processor as multiple processors. When you have a processor license, there are no restrictions on the number of client connections to the database server.
-o *filename*	Writes output messages to the named file.
-q	Runs in quiet mode—messages are not displayed.
-u *licensenumber*	Specifies the total number of users or processors for the license. If you are adding extra licenses, this is the total, *not* the number of additional licenses.
licensefile	Specifies the path and file name of the server executable or license file for the personal database server, network database server, or MobiLink server you are licensing. You can view the current license information for a server executable by entering only the license file name.
username	Specifies the user name for the license. This name appears in the database server messages window on startup. If there are spaces in the name, enclose it in double quotes.
companyname	Specifies the company name for the license. This name appears in the database server messages window on startup. If there are spaces in the name, enclose it in double quotes.

Remarks

The Server Licensing utility adds licensed users or licensed processors to your SQL Anywhere database server or MobiLink server. You must use this utility only in accordance with your license agreement to license the number of users or processors to which you are entitled. Running this command does not grant

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

you license. The number of CPUs that the database server can use may also be affected by your SQL Anywhere edition or the -gt server option. See:

- "Editions and licensing" [*SQL Anywhere 12 - Introduction*]
- "-gt dbeng12/dbsrv12 server option" on page 195

This utility also modifies the user and company names displayed at startup by the personal or network database servers, and the MobiLink server.

You can also use this utility to view the current license information for a personal or network database server by entering only the license file name.

Licensing information is stored in a *.lic* file in the same directory as the server executable. The server looks for a *.lic* file that has the same base file name as the executable that is being run. For example, if the database server executable was named *myserver.exe*, then the server looks for a license file named *myserver.lic*. By default, the following names are used:

Executable	License file name
SQL Anywhere personal database server (dbeng12)	*dbeng12.lic*
SQL Anywhere network database server (dbsrv12)	*dbsrv12.lic*
MobiLink server (mlsrv12)	*mlsrv12.lic*

When you attempt to start a server, if the corresponding *.lic* file is not available, then the server does not start. The license file is created by the SQL Anywhere installation program. The dblic utility only modifies existing licenses; it does not create new license files.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

On Unix, the database server executable is not writable by default, so using the Server Licensing (dblic) utility fails. Make sure the executable is writable (for example, using `chmod +w`) before you use the Server Licensing utility.

Cached connections count toward per seat licensing.

For more information about SQL Anywhere licensing, see http://www.sybase.com/detail?id=1056242.

Example

The following command, executed in the same directory as the database server executable, applies a license for 50 users, in the name of Sys Admin, for company My Co, to a Microsoft Windows network database server. The command must be entered all on one line:

```
dblic -l perseat -u 50 "c:\Program Files\SQL Anywhere 12\Bin32\dbsrv12.lic"
"Sys Admin" "My Co"
```

The following messages appear on the screen to indicate the success of the license:

```
SQL Anywhere Server Licensing Utility Version 12.0.0.2413
License applied successfully.
Advanced Edition
Add-on: High Availability
Add-on: In-Memory mode
Add-on: Scale-out Nodes
Licensed nodes: 50
User: Sys Admin
Company: My Co
Install key:
```

The following command returns information about the license for a database server:

```
dblic "c:\Program Files\SQL Anywhere 12\Bin32\dbsrv12.lic"
```

Service utility (dbsvc) for Linux

Creates, modifies, and deletes SQL Anywhere services.

Syntax

dbsvc [*modifier-options*] **-d** *svc*

dbsvc [*modifier-options*] **-g** *svc*

dbsvc [*modifier-options*] **-l**

dbsvc [*modifier-options*] **-status** *svc*

dbsvc [*modifier-options*] **-u** *svc*

dbsvc [*modifier-options*] *creation-options* **-w** *svc* [*options*]

dbsvc [*modifier-options*] **-x** *svc*

Major option	Description
-d *service-name*	Removes the named service from the list of services. If you supply -y, the service is deleted without confirmation.
-g *service-name*	Lists the definition of the service.
-l	Lists the available SQL Anywhere services.
-u *service-name*	Starts a service named *service-name*.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Major option	Description
-w *executable parameters*	Creates a new service, or overwrites one if one of the same name exists. If you supply -y, the existing service is overwritten without confirmation. You cannot delete a service while it is running. You must supply parameters appropriate for the service you are creating. See: • **dbsrv12 and dbeng12** "The SQL Anywhere database server" on page 147 • **mlsrv12** "MobiLink server options" [*MobiLink - Server Administration*] • **dbmlsync** "dbmlsync syntax" [*MobiLink - Client Administration*] • **dbremote** "SQL Remote Message Agent utility (dbremote)" [*SQL Remote*] • **dbns** "Broadcast Repeater utility (dbns12)" on page 772 • **rshost** "Relay Server State Manager" [*Relay Server*] • **rsoe** "Outbound Enabler" [*Relay Server*]
-x *service-name*	Stops a service named *service-name*.

Creation option	Description
-a *acct*	Names the account. All services run under a Linux account. If you run under an account you have created, you must name the account with the -a option.
-as	Runs the service under the Linux daemon account. All services run under a Linux account. No password is required. One of -a or -as must be used.
-od	Specifies the location of the system information file (if required).
-pr	Sets the nice level for the Linux process.
-rl	Specifies the runlevels on which to start the service.
-rs *service-name*	Adds the service to the list of services in the Required-Start header of the init script generated by dbsvc. This header helps the operating system determine the order in which it starts services. Specified service names are verified to see if they already exist. The naming convention for the service is **SA_***service-name*.

Creation option	Description
-s	Sets the startup behavior for SQL Anywhere services. You can set startup behavior to Automatic or Manual. The default is Manual.
-status	Returns the state the service is running.
-t *type*	Specifies the type for this service. You can choose from the following types (the alternate name is listed in parentheses): • **Network** Creates a service for the SQL Anywhere network database server (dbsrv12). See "The SQL Anywhere database server" on page 147. • **Personal (Standalone)** This is the default, and creates a service for the SQL Anywhere personal database server (dbeng12). See "The SQL Anywhere database server" on page 147. • **DBRemote** Creates a service for the SQL Remote Message Agent. See "SQL Remote Message Agent utility (dbremote)" [*SQL Remote*]. • **MobiLink** Creates a service for the MobiLink server (mlsrv12). See "mlsrv12 syntax" [*MobiLink - Server Administration*]. • **DBMLsync (MLsync)** Creates a service for the MobiLink synchronization client (dbmlsync). See "dbmlsync syntax" [*MobiLink - Client Administration*]. • **dbns** Creates a service for the Broadcast Repeater utility (dbns12). See "Broadcast Repeater utility (dbns12)" on page 772. • **rshost** Creates a service for the Relay Server State Manager. See "Relay Server State Manager command line syntax" [*Relay Server*]. • **rsoe** Creates a service for the Outbound Enabler. See "Outbound Enabler syntax" [*Relay Server*].

Modifier option	Description
-cm	Displays the command used to create the service. This option can be used to output the creation command to a file, which can then be used to add the service on another computer or restore a service to its original state if changes have been made to it. You must specify the -g option or -l option with -cm or the command fails. Specifying -g displays the creation command for the specified service, while specifying -l displays the creation command for all services.
-q	Suppresses messages to the database server messages window. If you specify this option when modifying or deleting an existing service, you must also specify -y or the operation fails.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Modifier option	Description
-y	Automatically performs the action without prompting for confirmation. This option can be used with the -w or -d options. If you specify -q when modifying or deleting an existing service, you must also specify -y or the operation will fail.

Remarks

A service runs a database server or other application with a set of options. This utility provides a comprehensive way of managing SQL Anywhere services on Linux.

Because services typically run in a different environment, it is recommended that you fully qualify the name of the database file when creating a service. It is also recommended that you do not use spaces in data source names.

You must have permissions on the */etc/init.d* directory to create, edit, or delete services.

When a SQL Anywhere service is running on Linux, a PID file is created in the */var/run* directory. This file contains the PID of the dbsvc process. The file is named ***SA_***_service-name_***.pid***. This file can be used by other Linux tools to find the process and monitor the service.

Like most Linux services, the dbsvc utility creates service files in */etc/init.d*. The naming convention for the service is **SA_***service-name*. For example, if you created a service named myserv, you could issue the following command to start the service:

```
/etc/init.d/SA_myserv start
```

The following command gets the status of the service:

```
/etc/init.d/SA_myserv status
```

The following command returns usage information for the service:

```
/etc/init.d/SA_myserv
```

Note
If your dbsvc command contains characters that are special to the shell (such as semi-colons or parentheses), it is recommended that you use a configuration file for the parameters to your service. See "Using configuration files" on page 763.

Example

Create a personal server service called myserv, which starts the specified server with the specified parameters. The server runs as the LocalSystem user:

```
dbsvc -as -w myserv -n myeng -c 8m "/tmp/demo.db"
```

Create a network server service called mynetworkserv. The server runs under the local account, and starts automatically when the computer is restarted:

```
dbsvc -as -t network -w mynetworkserv -x tcpip -c 8m "/tmp/demo.db"
```

List all details about service myserv:

```
dbsvc -g myserv
```

Delete the service called myserv, without prompting for confirmation:

```
dbsvc -y -d myserv
```

Create a service called mysyncservice:

```
dbsvc -as -t dbmlsync -w mysyncservice -c "/tmp/CustDB.db" -o syncinfo.txt
```

Generate the command to create the service_1 service and output it the console:

```
dbsvc -cm -g service_1
```

The console contains the following:

```
'dbsvc -t Standalone -as -y -w "service_1" -n'
```

Start a service using dbsvc:

```
dbsvc -u myserv
```

Use dbsvc to stop a service:

```
dbsvc -x myserv
```

Use dbsvc to obtain the status of a service:

```
dbsvc -status myserv
```

Service utility (dbsvc) for Windows

Creates, modifies, and deletes SQL Anywhere services.

Syntax

dbsvc [*modifier-options*] **-d** *svc*

dbsvc [*modifier-options*] **-g** *svc*

dbsvc [*modifier-options*] **-l**

dbsvc [*modifier-options*] **-u** *svc*

dbsvc [*modifier-options*] *creation-options* **-w** *svc details*

dbsvc [*modifier-options*] **-x** *svc*

details:

<*full-executable-path*> [*options*]

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Major option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-d *service-name*	Removes the named service from the list of services. If you supply -y, the service is deleted without confirmation.
-g *service-name*	Lists the definition of the service, not including the password.
-l	Lists the available SQL Anywhere services.
-u *service-name*	Starts the service named *service-name*.

Major option	Description
-w *executable parameters*	Creates a new service, or overwrites one if one of the same name exists. If you supply -y, the existing service is overwritten without confirmation. You cannot delete a service while it is running. You must supply the full path to the executable that you want to use as a service, as the account under which the service is running may not have the appropriate SQL Anywhere installation directory in its path. You must supply parameters appropriate for the service you are creating. See: • **dbsrv12 and dbeng12** "The SQL Anywhere database server" on page 147 • **mlsrv12** "MobiLink server options" [*MobiLink - Server Administration*] • **dbmlsync** "dbmlsync syntax" [*MobiLink - Client Administration*] • **dblsn** "MobiLink Listener utility for Windows devices (dblsn)" [*MobiLink - Server-Initiated Synchronization*] • **dbremote** "SQL Remote Message Agent utility (dbremote)" [*SQL Remote*] • **dbns** "Broadcast Repeater utility (dbns12)" on page 772 • **rshost** "Relay Server State Manager" [*Relay Server*] • **rsoe** "Outbound Enabler" [*Relay Server*] • **dbvss12** "Using the SQL Anywhere Volume Shadow Copy Service (VSS)" on page 898
-x *service-name*	Stops the service named *service-name*.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Creation option	Description
-a *acct*	Names the Microsoft Windows account. All services run under a Microsoft Windows account. If you run under an account you have created, you must name the account with the -a option and supply a password with the -p option. The Login as a Service privilege is required for all accounts other than the LocalSystem account. If an account does not have the Login as a Service privilege enabled, you are prompted to enable it. If the -y option is also specified, dbsvc attempts to grant the Login as a Service privilege without prompting you. If the -q option is specified without the -y option, you are not prompted to grant the Login as a Service privilege and dbsvc fails.
-as	Runs the service under the Microsoft Windows LocalSystem account. No password is required. One of -a or -as must be used. All services run under a Microsoft Windows account.
-i	Displays an icon that you can double-click to display the database server messages window.
-p	Specifies the password for the account under which the service runs. Use this option with the -a option.
-rg *dependency,...*	Specifies one or more load ordering groups that must be started before the service being created is allowed to start.

Creation option	Description
-rs *dependency*,...	Specifies that all the services in the list must have started before the service being created is allowed to start. You can specify the display name or the service name. Service names are verified to see if they already exist. If the specified service name cannot be found, the Service utility (dbsvc) checks to see if there is a service with a matching display name. It is recommended that you create a dependency on the Transport Data Interface (TDI) for TCP/IP to ensure that the connection starts correctly. For more information about the TDI, see http://msdn.microsoft.com/en-us/library/ms797340.aspx.
-s *startup*	Sets startup behavior for SQL Anywhere services. You can set startup behavior to Automatic, Manual, or Disabled. The default is Manual.
-sd *description*	Provides a description of the service. The description appears in the Windows Service Manager.

-sn *name*	Provides a name for the service. This name appears in the Windows Service Manager. If you do not specify the -sn option, the default service name is SQL Anywhere - *svc*. For example, the following service is named SQL Anywhere - myserv by default.
	
	`dbsvc -as -w myserv "C:\Program Files\SQL Anywhere 12\Bin32\dbeng12.exe"`
	
	To have the service name myserv appear in the Windows Service Manager, you need to execute the following (entered all on one line):
	
	`dbsvc -as -sn myserv -w myserv "C:\Program Files\SQL Anywhere 12\Bin32\dbeng12.exe"`

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Creation option	Description
-t *type*	Specifies the type for this service. You can choose from the following types: • **Network** Creates a service for the SQL Anywhere network database server (dbsrv12). See "The SQL Anywhere database server" on page 147. • **Personal (Standalone)** This is the default, and creates a service for the SQL Anywhere personal database server (dbeng12). See "The SQL Anywhere database server" on page 147. • **DBRemote** Creates a service for the SQL Remote Message Agent. See "SQL Remote Message Agent utility (dbremote)" [*SQL Remote*]. • **MobiLink** Creates a service for the MobiLink server (mlsrv12). See "mlsrv12 syntax" [*MobiLink - Server Administration*]. • **DBMLsync (MLsync)** Creates a service for the MobiLink synchronization client (dbmlsync). See "dbmlsync syntax" [*MobiLink - Client Administration*]. • **dbns** Creates a service for the Broadcast Repeater utility (dbns12). See "Broadcast Repeater utility (dbns12)" on page 772. • **dblsn (LSN)** Creates a service for the MobiLink Listener utility. See "MobiLink Listener utility for Windows devices (dblsn)" [*MobiLink - Server-Initiated Synchronization*]. • **dbvss (VSS)** Creates a service for the SQL Anywhere VSS writer. "Using the SQL Anywhere Volume Shadow Copy Service (VSS)" on page 898 • **rshost** Creates a service for the Relay Server State Manager. See "Relay Server State Manager command line syntax" [*Relay Server*].

Creation option	Description
	• **rsoe** Creates a service for the Outbound Enabler. See "Outbound Enabler syntax" [*Relay Server*].

Modifier option	Description
-cm	Displays the command used to create the service. This option can be used to output the creation command to a file, which can then be used to add the service on another computer or restore a service to its original state if changes have been made to it. You must specify the -g option or -l option with -cm or the command fails. Specifying -g displays the creation command for the specified service, while specifying -l displays the creation command for all services. If the specified service does not exist, the command to delete the service is generated. For example, if service_1 does not exist on the computer, `dbsvc -cm -g service_1` would return the following command to delete the service_1 service: `dbsvc -y -d "service_1"` If the service does not use the LocalSystem account, there is no way to retrieve the password, so it is not included in the command that is generated. If you created the service with `-a user -p password`, only `-a user` is included in the output.
-o *logfile*	Writes output from the Service utility (dbsvc) to the specified file. The -o option must be specified before the -d, -g, -l, -u, and -x options. When you specify the -o option for both dbsvc and for the executable that you are running as a service (for example, the database server), log files are created for both. For example: `dbsvc -o out1.txt -y -as -w mydsn install-dir\bin32\dbsrv12 -n mysrv -o c:\out2.txt` In this case, the output from dbsvc is logged to *out1.txt*, while the output from the database server is logged to *c:\out2.txt*.
-q	Suppresses messages to the database server messages window. If you specify -q, it is also recommended that you specify a file where messages are logged using the -o option. If you specify this option when modifying or deleting an existing service, you must also specify -y or the operation will fail.
-y	Automatically performs the action without prompting for confirmation. This option can be used with the -w or -d options. If you specify -q when modifying or deleting an existing service, you must also specify -y or the operation will fail.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

A service runs a database server or other application with a set of options. This utility provides a comprehensive way of managing SQL Anywhere services on Windows. You must be a member of the Administrators group on the local computer to use the Service utility.

You can access the Service utility in the following ways:

- From Sybase Central, using the **Create Service Wizard**. See "Create Windows services" on page 59.
- At a command prompt, using the dbsvc command.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

See also

- "Understanding Windows services" on page 58

Example

Create a personal server service called myserv, which starts the specified server with the specified parameters. The server runs as the LocalSystem user:

```
dbsvc -as -w myserv
"C:\Program Files\SQL Anywhere 12\Bin32\dbeng12.exe"
-n myeng -c 8m "c:\temp\mysample.db"
```

Create a network server service called mynetworkserv. The server runs under the local account, and starts automatically when the computer is restarted:

```
dbsvc -as -s auto -t network -w mynetworkserv
"C:\Program Files\SQL Anywhere 12\Bin32\dbsrv12.exe"
-x tcpip -c 8m "c:\temp\mysample.db"
```

List all details about service myserv:

```
dbsvc -g myserv
```

Delete the service called myserv, without prompting for confirmation:

```
dbsvc -y -d myserv
```

Create a service dependent on the Workstation service and the TDI group:

```
dbsvc -rs lanmanworkstation -rg TDI -w ...
```

Create a service called mysyncservice:

```
dbsvc -as -s manual -t dbmlsync -w mysyncservice
"C:\Program Files\SQL Anywhere 12\Bin32\dbmlsync.exe"
-c "SQL Anywhere 12 CustDB"
```

Generate the command to create the service_1 service and output it to a file called *restoreservice.bat*:

```
dbsvc -cm -g service_1 > restoreservice.bat
```

The *restoreservice.bat* file contains the following:

```
dbsvc -t Standalone -s Manual -as -y -w "service_1"
"C:\Program Files\SQL Anywhere 12\Bin32\dbeng12.exe"
```

Create a MobiLink listener service that is started manually:

```
dbsvc -as -i -w myListener
"C:\Program Files\SQL Anywhere 12\Bin32\dblsn.exe" "@c:\temp\dblsn.opt"
```

Start the myListener service:

```
dbsvc -u myListener
```

Stop the myListener service:

```
dbsvc -x myListener
```

Create a Volume Shadow Copy Service (VSS) service that is started automatically when the database server starts:

```
dbsvc -as -s Automatic -t vss -w SAVSSWriter
"C:\Program Files\SQL Anywhere 12\Bin32\dbvss12.exe"
```

SQL Anywhere Console utility (dbconsole)

Provides administration and monitoring facilities for database server connections.

Syntax

dbconsole [*options*]

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-c *"keyword=value; ..."*	Specifies connection parameters. See "Connection parameters" on page 265.
-datasource *DSN-name*	Specifies an ODBC data source to connect to. You do not need to be using the SQL Anywhere JDBC driver to use this option.
-host *hostname*	Specifies the *hostname* or IP address of the computer on which the database server is running. You can use the name localhost to represent the current computer.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-port *port-number*	Specifies the port number on which the database server is running. The default port number for SQL Anywhere is **2638**.

Remarks

The SQL Anywhere Console utility allows you to monitor the server from a client computer. This utility is also called the Network Server Monitor. You can use it to track who is logged on to a database server elsewhere on your network. You can also display both database server and client statistics on your local client screen, disconnect users, and configure the database server. The SQL Anywhere Console can display information for multiple connections.

The SQL Anywhere Console is available on all supported platforms except Windows Mobile, IBM AIX, HP-UX, and HP-UX Itanium. On these platforms, you can use the connection-level, server-level, and database-level properties to obtain information or you can monitor your server from a computer running an operating system that supports the SQL Anywhere Console (such as Windows, Mac OS X, or Linux).

For more information about obtaining property values, see "Connection, database, and database server properties" on page 619.

See also

- "Using the SQL Anywhere Console utility" on page 759

Start Server in Background utility (dbspawn)

Starts a database server in the background.

Syntax

dbspawn [*options*] *server-command*

Option	Description

@data	Reads in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.
	
	The Start Server in Background utility (dbspawn) allows you to specify a configuration file reference in the command to be spawned, but you cannot specify a configuration file with options for the dbspawn utility. For example, the first command below is supported, but the second command is *not* supported:
	
	`dbspawn dbeng12 @myconfig.ini`
	
	`dbspawn @spawnopts.ini dbeng12 demo.db`
	
	For more information, see "Using configuration files" on page 763.
	
	If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.

-f	Forces dbspawn to start a database server, even if a default database server already exists. If a database server is running but is not the default, dbspawn starts another server. If a database server is already running with the same name as the database server that dbspawn is attempting to start, dbspawn returns success without starting a new server.

-p	Specifies the operating system process ID of the database server process. For example:
	
	`dbspawn -p dbeng12 -n newserver`
	
	reports a message of the following form to a command prompt:
	
	`New process ID is 306`

-q	Runs in quiet mode—messages are not displayed.
server-command	Specifies the command line for starting the database server. See "The SQL Anywhere database server" on page 147.

Remarks

The dbspawn utility is provided to start a server in the background. dbspawn starts the server in the background and returns with an exit code of 0 (success) or non-zero (failure). If a database server is already running on the same computer, dbspawn does not start the new server and reports failure. Otherwise, dbspawn does not return until the database server has completed initialization and is ready to accept requests.

For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The dbspawn utility is useful for starting a server from a batch file, especially when subsequent commands in the batch file require a server that is accepting requests.

If the specified path includes at least one space, you must enclose the path in one set of double quotes. For example,

```
dbspawn dbeng12 "c:\my databases\mysalesdata.db"
```

If the specified path does not contain spaces, then quotes are not required.

Stop Server utility (dbstop)

Stops a database or database server.

Syntax

dbstop [*options*] [*server-name*]

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-c *"keyword=value; ..."*	Specifies a connection string. When stopping a network server, the connection string must include a user ID that has permissions to stop the server. By default, DBA authority is required on the network server, and all users can shut down a personal server, but the -gk server option can be used to change this. If you supply connection parameters, do not supply a server name as well. See "Connection parameters" on page 265, "Unconditional (UNC) connection parameter" on page 309, and "-gk dbeng12/dbsrv12 server option" on page 187.
-d	Does not stop the database server. Instead, only stop the database specified in the connection string. The -gd server option controls the permissions required to stop a database file. See "-gd dbeng12/dbsrv12 server option" on page 185.
-o *filename*	Writes output messages to the named file.
-q	Runs in quiet mode—messages are not displayed.
-x	Does not stop the server if there are still active connections to the server. Including this option prevents dbstop from prompting for confirmation if there are active connections.

Option	Description
-y	Stops the server even if there are still active connections to the server. This is equivalent to including Unconditional=YES in the connection parameters.
server-name	Specifies the name of a database server running on the current computer. The database server must be started so that no permissions are required to shut it down. The personal database server starts in this mode by default. For the network database server, you must supply the **-gk all** option. See "-gk dbeng12/dbsrv12 server option" on page 187. If you supply a server name, do not supply connection parameters as well.

Remarks

The Stop Server utility stops a database server. You can use the -d option to stop a specified database.

The Stop Server utility can only be run at a command prompt. In windowed environments, you can stop a database server by clicking **Shut Down** on the database server messages window.

Options let you control whether a server is stopped, even if there are active connections, and whether to stop a server or only a database.

The behavior of dbstop can be controlled if there are active connections on a server. If there are active connections, dbstop provides a prompt asking if you want to shut down the server. The -x and -y options can be used to change this behavior.

If dbstop is able to stop the database server, dbstop does not complete until all databases have stopped running, and the database server has been stopped enough so that another server could be started with the same name and databases. When dbstop successfully completes, the database server process may still be running, and some of its resources, such as the output file specified by the -o server option, may still be in use.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

If you want to use the SQLCONNECT environment variable with dbstop, you should specify the -c option. Otherwise, you can get unexpected results.

See also

- "STOP SERVER statement" [*SQL Anywhere Server - SQL Reference*]

Example

You are running the server named myserver without a database. To stop the server, specify the utility database as a DatabaseName (DBN) connection parameter:

```
dbstop -c "UID=DBA;PWD=sql;Server=myserver;DBN=utility_db"
```

You are running the server named myserver with the database *demo.db* started. To stop the server and database:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
dbstop -c "UID=DBA;PWD=sql;Server=myserver"
```

You are running a personal server named myserver. To stop the server and databases even if there are connections:

```
dbstop -y myserver
```

You are running a server named myserver with the database *demo.db*. To stop only the database named demo, but not other databases or the server itself, execute the following command:

```
dbstop -c "UID=DBA;PWD=sql;Server=myserver;DBN=demo" -d
```

Support utility (dbsupport)

Sends information about errors and software usage to iAnywhere Solutions.

Syntax

dbsupport [*options*] *operation* [*operation-specific-option*]

dbsupport *configuration-options*

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-o *filename*	Sends output to the specified file.
-q	Displays only critical messages.

Operation	Description

-e *configuration-option*	Displays the setting for the specified configuration option. For example, suppose you ran the following command to configure dbsupport to prompt when possible:
	
	`dbsupport -cc promptdefy`
	
	When you run the command `dbsupport -ecc`, the following setting is returned:
	
	`-cc "promptdefy"`

Operation	Description
-is *submission-ID* [**-rr** *N*]	Checks the status of a crash report that has been submitted to iAnywhere Solutions. For example, the following command inquires about the status of submission ID 66: `dbsupport -is 66`
-iu [**-r** *N*]	Checks for updates to your build of SQL Anywhere. Updates include: • **EBF** An express bug fix is a subset of the software with one or more bug fixes. The bug fixes are listed in the release notes for the update. Bug fix updates may only be applied to installed software with the same version number. While some testing has been performed on the software, you should not distribute these files with your application unless you have thoroughly tested your application with the software. • **Maintenance releases** A maintenance release is a complete set of software that upgrades installed software from an older version with the same major version number (version number format is *major.minor.patch.build*). Bug fixes and other changes are listed in the release notes for the upgrade. You can also check for updates using Interactive SQL and Sybase Central. See "Checking for software updates" on page 762.
-lc	Generates a list of all crash reports that have not been submitted to iAnywhere Solutions. The report names listed can be used with the -sc option.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Operation	Description

-ls	Generates a list of submission IDs for all reports that have been submitted to iAnywhere Solutions. For example:
	`dbsupport -ls`
	This returns information similar to the following:
	`SQL Anywhere Support Utility Version 12.0.0.2040`
	`Submission ID: 217756  reported: Wed Jul 30 15:54:43 2008`
	`Submission ID: 217757  reported: Wed Jul 30 15:57:07 2008`
	`Submission ID: 217759  reported: Wed Jul 30 16:14:24 2008`
	`Submission ID: 217760  reported: Wed Jul 30 16:14:30 2008`
	`Submission ID: 217761  reported: Wed Jul 30 16:28:12 2008`
	`Submission ID: 221030  reported: Mon Sep 08 12:34:59 2008`
	`Submission ID: 221031  reported: Mon Sep 08 12:35:05 2008`

-pc *filename*	Displays crash report information. You can use this option to view information before it is submitted to iAnywhere Solutions.
-pd	Displays the diagnostic information that has been collected. You can use this option to view information before it is submitted to iAnywhere Solutions.

-ps *submission-ID*	Displays information about a specific report that has been submitted to iAnywhere Solutions. For example:
	`dbsupport -ps 4`
	This returns information about submission 4:
	`SQL Anywhere Support Utility Version 12.0.0.2040`
	`Submission ID: 217756  reported: Wed Jul 30 15:54:43 2008`

-sa [**-r** *number-of-submission-retries*]	Submits all crash report and diagnostic information stored in the diagnostic directory to iAnywhere Solutions.

Operation	Description
-sc *reportname* [**-r** *number-of-submission-retries*] [**-nr** | **-rr** *N*]	Submits a crash report and diagnostic information to iAnywhere Solutions. For example: `dbsupport -sc SA12_20051220_133828_32116` Use the -lc option to see a list of reports that have not been submitted.
-sd [**-r** *number-of-submission-retries*]	Submits only diagnostic information to iAnywhere Solutions. For more information about the diagnostic directory, see "SADIAGDIR environment variable" on page 382.

Configuration option	Description
-cc [**autosubmit** | **no** | **promptDefY** | **promptDefN**]	Changes the prompting behavior of dbsupport. You can specify one of the following options: • **autosubmit** Submit reports automatically. • **no** Do not prompt for permission to submit reports. Reports and feature statistics are not submitted. • **promptDefY** If possible, prompt for permission to submit the report. If no answer is given, submit the report. • **promptDefN** If possible, prompt for permission to submit the report. If no answer is given, do not submit the report. This is the default behavior. For example, if you are using embedded SQL Anywhere in an application, you may want to configure the Support utility to not submit reports to iAnywhere Solutions. If you specify this option, its value becomes the default used by the Support utility. The setting is stored in the *dbsupport.ini* file in the diagnostic directory. The following command configures the Support utility so that it does not submit reports and never prompts the user to submit reports: `dbsupport -cc no`

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Configuration option	Description
-cd *retry-delay*	Specifies the retry delay, in seconds, for submitting a report if the previous attempt is unsuccessful. The default delay is 30 seconds. If you specify this option, its value becomes the default used by the Support utility. The setting is stored in the *dbsupport.ini* file in the diagnostic directory. The following dbsupport command specifies that failed submissions should be retried every 3 seconds, up to a maximum of 4 times before giving up: `dbsupport -cr 4 -cd 3`
-ce *email-address***;***email-server*[**:***port*][**;***user-id***:***password*]	Specifies the address where an email is sent after a crash occurs. The email is sent using the *email-server* SMTP server. Optionally, you can specify the port that should be used, and the *user-id* and *password* used to authenticate with the SMTP server.
-cet	Test the email settings specified by the -ce option.

Configuration option	Description

-ch *crash-handler-program*	Specifies a program that is called when a crash occurs.
	
	If you specify this option, its value becomes the default used by the Support utility. The setting is stored in the *dbsupport.ini* file in the diagnostic directory.
	
	The database server supports three substitution parameters that set up information that is passed to the *crash-handler-program*:
	
	● **%F** This parameter is replaced with the full path to the location of the generated report file.
	
	● **%P** This parameter is replaced with the name of the program that generated the report. For example, if a version 12 personal database server generates the report, dbeng12 is returned.
	
	● **%S** This parameter is replaced with the name of the database server that was in use when the crash or fatal error occurred. For example, if a database server named Sample generated the report, Sample is returned.
	
	You can use $F, $P, and $S as alternatives to %F, %P, and %S. Because different command shells interpret the characters % and $, both are provided. For example, on 4NT, %F is substituted with the value of the environment variable F; $F can be used to avoid this substitution.
	
	Suppose you have a crash handler program in *c:\test.bat* that contains the following commands:
	
	`copy %1 c:\archives`
	`echo %2`
	
	On Windows, the following command tells dbsupport to launch *c:\test.bat* with two parameters when a crash occurs. If the report is being submitted, this program is called before the report is submitted.
	
	`dbsupport -ch "c:\test.bat \"%F\" parm2"`
	
	The substituted path specified by %F is sent to *c:\test.bat* as the first parameter. The parameter *parm2* is sent to *c:\test.bat* as the second parameter. Note

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Configuration option	Description
	that quotation marks must be used to specify a crash handler program that takes arguments. In the example above, additional quotes were used around the full path to the generated report file. To avoid problems accessing the report file that dbsupport is using, the crash handler program should make its own copy of the report file.
-ch-	Removes the crash handler settings that are stored in the *dbsupport.ini* file. For example: `dbsupport -ch-`

-cid *customer-id*	Specifies a string that identifies you in the submission report. If you specify this option, its value becomes the default used by the Support utility. The configuration is stored in the *dbsupport.ini* file in the diagnostic directory.
	
	The following examples specify a customer identification string for dbsupport:
	
	`dbsupport -cid myid@company.com`
	
	`dbsupport -cid "MyClientApp 1.0"`

-cid-	Removes the customer identification string from the *dbsupport.ini* file. For example: `dbsupport -cid-`
-cp { *email-server* [**:***port*] | **autodetect** }	Configures the HTTP proxy host and port used to submit error reports. On Windows, the syntax -cp autodetect is also supported. If you specify this option, then if a proxy server and port have been set using Internet Explorer, and they are currently enabled, dbsupport configures its proxy server and port using the system setting. You can set the proxy server and port in Internet Explorer on Windows by choosing **Tools** » **Options** » **Lan Settings**.
-cp-	Removes HTTP proxy host and port settings from the *dbsupport.ini* file. For example: `dbsupport -cp-`

Configuration option	Description
-cr *number-of-submission-retries*	Specifies the number of times a failed submission should be retried. If you specify this option, its value becomes the default used by the Support utility. The setting is stored in the *dbsupport.ini* file in the diagnostic directory. The following dbsupport command specifies that failed operations should be retried every 3 seconds, up to a maximum of 4 times before giving up: `dbsupport -cr 4 -cd 3`

Operation-specific option	Description
-ac	Adds a comment to the submission.
-af	Attaches a file to the submission. You can specify the -af option more than once to attach multiple files to the submission.
-nr	Specifies that dbsupport does not check the server for the status of the submission. For example, the following command submits the report, but does not check for the status of the new submission: `dbsupport -nr -sc SA12_20051220_133828_32116` By default, dbsupport checks whether there is already a fix for the problem being submitted.
-r *number-of-submission-retries*	Specifies the maximum number of times dbsupport should try to send the submission. Specifying 0 means retry indefinitely. The default value is 10. Specifying -r overrides the -cr value stored in the *dbsupport.ini* file, if one exists.
-rd *retry-delay*	Specifies the number of seconds dbsupport waits between attempts to resend the report. The default value is 30. Specifying -rd overrides the -cd value stored in the *dbsupport.ini* file, if one exists.
-rr *number-of-submission-response-retries*	Specifies the maximum number of times dbsupport should try to obtain a submission response. Specifying 0 means retry indefinitely. The default value is 10.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

The Support utility (dbsupport) can be used for any of the following tasks:

- submit diagnostic information and crash reports to iAnywhere Solutions over the Internet
- submit feature statistics
- list information about submitted and unsubmitted crash reports
- print information about submitted and unsubmitted crash reports
- inquire about the status of a submission
- inquire whether there are software updates available for your build of SQL Anywhere
- configure what is to be done when a fatal error (assertion/crash) is detected by a database or MobiLink server

By default, dbsupport checks whether there is already a fix for the problem being submitted.

Information from any of the following applications can be sent as an error report if a fatal error occurs:

- Interactive SQL (dbisql)
- MobiLink Listener (dblsn)
- MobiLink replay API generator (mlgenreplayapi)
- MobiLink replay utility (mlreplay)
- MobiLink server (mlsrv)
- network server (dbsrv12)
- personal server (dbeng12)
- QAnywhere agent (qaagent)
- SQL Anywhere client for MobiLink (dbmlsync)
- SQL Anywhere Console utility (dbconsole)
- SQL Remote (dbremote)
- Sybase Central

When a report is successfully submitted, it is assigned a unique submission ID. Reports are written to the diagnostic directory.

Platform	Default diagnostic directory location
Windows (except Windows Mobile)	*%ALLUSERSPROFILE%\Application Data\SQL Anywhere 12\diagnostics*
Windows Mobile	Directory where the executable is running.
Unix	*$HOME/.sqlanywhere12/diagnostics*

For information about the diagnostic directory, see "SADIAGDIR environment variable" on page 382.

For more information about error reports and how they are submitted, see "Error reporting in SQL Anywhere" on page 996.

The Support utility can also be configured to perform certain actions when a problem is detected. For example, it can be configured to execute a specified handler program each time the database server submits an error report. This feature is useful for adding your own custom actions to the error handling process.

As well, the Support utility can be configured to retry certain operations. For example, when submitting a report, it could be configured to retry the operation again in 30 seconds, up to a maximum of 10 times. This feature is useful for handling the case where the service may be temporarily unavailable.

Settings for the Support utility are stored in the *dbsupport.ini* file in the diagnostic directory.

The operation-specific options are useful for overriding default behavior, including those that have been saved in the *dbsupport.ini file*.

- **EBF** An express bug fix is a subset of the software with one or more bug fixes. The bug fixes are listed in the release notes for the update. Bug fix updates may only be applied to installed software with the same version number. While some testing has been performed on the software, you should not distribute these files with your application unless you have thoroughly tested your application with the software.

- **Maintenance release** A maintenance release is a complete set of software that upgrades installed software from an older version with the same major version number (version number format is *major.minor.patch.build*). Bug fixes and other changes are listed in the release notes for the upgrade.

See also

- "SADIAGDIR environment variable" on page 382
- "Software component exit codes" [*SQL Anywhere Server - Programming*]

Transaction Log utility (dblog)

Administers the transaction log for a database.

Syntax

dblog [*options*] *database-file*

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-ek *key*	Specifies the encryption key for strongly encrypted databases directly in the command. If you have a strongly encrypted database, you must provide the encryption key to use the database or transaction log in any way. For strongly encrypted databases, you must specify either -ek or -ep, but not both. The command will fail if you do not specify the correct key for a strongly encrypted database.
-ep	Specifies that you want to be prompted for the encryption key. This option causes a window to appear, in which you enter the encryption key. It provides an extra measure of security by never allowing the encryption key to be seen in clear text. For strongly encrypted databases, you must specify either -ek or -ep, but not both. The command will fail if you do not specify the correct key for a strongly encrypted database.
-ir	Resets the SQL Remote log offset that is kept for the delete_old_logs option, allowing transaction logs to be deleted when they are no longer needed. Use this option if you have stopped using SQL Remote on this database, but continue to use MobiLink synchronization.
-is	Resets the MobiLink log offset that is kept for the delete_old_logs option, allowing transaction logs to be deleted when they are no longer needed. Use this option if you have stopped using MobiLink synchronization on this database, but continue to use SQL Remote.
-m *mirror-name*	Specifies the file name for a new transaction log mirror. If the database is not currently using a transaction log mirror, it starts using one. If the database is already using a transaction log mirror, it changes to using the new file as its transaction log mirror.
-n	Stops using a transaction log, and stops using a transaction log mirror. Without a transaction log, the database can no longer participate in data replication or use the transaction log in data recovery. If a SQL Remote or dbmlsync truncation offset exists, the transaction log cannot be removed unless the corresponding ignore option (-ir for SQL Remote, or -is for dbmlsync) is also specified. You cannot stop using a transaction log if the database has auditing turned on (unless you first turn auditing off).
-o *filename*	Writes output messages to the named file.
-q	Runs in quiet mode—messages are not displayed.
-r	Maintains a single transaction log for databases that maintains a transaction log mirror.
-t *log-name*	Specifies the file name for a new transaction log. If the database is not currently using a transaction log, it starts using one. If the database is already using a transaction log, it changes to using the new file as its transaction log.
-x *n*	Resets the transaction log current relative offset to *n*, so that the database can take part in replication. This option is used for reloading SQL Remote consolidated databases. See "Extracting remote databases to a reload file" [*SQL Remote*].

Option	Description
-z *n*	Resets the transaction log starting offset to *n*, so that the database can take part in replication. This option is used for reloading SQL Remote consolidated databases. See "Extracting remote databases to a reload file" [*SQL Remote*].

Remarks

The dblog utility allows you to display or change the name of the transaction log or transaction log mirror associated with a database. You can also stop a database from maintaining a transaction log or mirror, or start maintaining a transaction log or mirror.

A transaction log mirror is a duplicate copy of a transaction log, maintained by the database in tandem.

The name of the transaction log is first set when the database is initialized. The Transaction Log utility works with database files. The database server must not be running on that database when the transaction log file name is changed (or an error message appears).

The utility displays additional information about the transaction log, including the following:

- Version number
- The name of the transaction log file
- The name of the transaction log mirror file, if any
- The current relative offset

You can access the Transaction Log utility in the following ways:

- From Sybase Central, using the **Change Log File Settings Wizard**. See "Change the location of a transaction log" on page 23.
- From Interactive SQL, using the ALTER DATABASE *dbfile* ALTER LOG statement. See "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*].
- At a command prompt, using the dblog command.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

Unload utility (dbunload)

Unloads a database into a SQL command file.

Syntax

dbunload [*options*] [*directory*]

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.

-ac *"keyword=value; ..."*	Connects to an existing database and reload the data directly into it, combining the operations of unloading a database and reloading the results into an existing database. This option is not supported on Windows Mobile.
	
	For example, you could create a new database using the Initialization utility, and then reload it using this option. This method is useful when you want to change initialization options.
	
	The following command (entered all on one line) loads a copy of the *c:\mydata.db* database into an existing database file named *c:\mynewdata.db*:
	
	`dbunload -c "UID=DBA;PWD=sql;DBF=c:\mydata.db" -ac "UID=DBA;PWD=sql;DBF=c:\mynewdata.db"`
	
	If the original database was created using version 9 or earlier of SQL Anywhere and the new database is not already running, you must provide a database server name in the -ac option. For example:
	
	`dbunload -c "UID=DBA;PWD=sql;DBF=c:\mydata.db" -ac "UID=DBA;PWD=sql;DBF=c:\mynewdata.db;Server=newserver"`
	
	If you use this option, no interim copy of the data is created on disk, so do not specify an unload directory in the command. This provides greater security for your data.

Option	Description

-an *database*	Combines the operations of unloading a database, creating a new database, and loading the data using this option. This option is not supported on Windows Mobile or when rebuilding version 9 or earlier databases on Mac OS X on Intel.
	
	The options specified when you created the source database are used to create the new database. However, you can change the initialization options as necessary by specifying other supported dbunload options (such as -ap to change the page size or -et to enable table encryption).
	
	For example, the following command (which should be entered all on one line) creates a new database file named *mydatacopy.db* and copies the schema and data of *mydata.db* into it:
	
	`dbunload -c "UID=DBA;PWD=sql;DBF=c:\mydata.db"`
	`-an c:\mydatacopy.db`
	
	If you use this option, no interim copy of the data is created on disk, so you do not specify an unload directory in the command. This provides greater security for your data.
	
	When the new database is created, the dbspace file names have an R appended to the file name to prevent file name conflicts if the dbspace file for the new database is created in the same directory as the dbspace for the original database. For example, if an unloaded database has a dbspace called library in the file *library.db*, then the library dbspace for the new database is *library.dbR*.
	
	The file specified by -an is relative to the database server.

-ap *size* [**k**]	Sets the page size of the new database. This option is ignored unless -an or -ar is also used. The page size for a database can be (in bytes) 2048, 4096, 8192, 16384, or 32768, with the default being the page size of the original database. Use **k** to specify units of kilobytes (for example, -ap 4k). You must specify either -an or -ar with this option. If there are already databases running on the database server, the server's page size (set with the -gp option) must be large enough to handle the new page size. See "-gp dbeng12/dbsrv12 server option" on page 193.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-ar [*directory*]	Creates a new database with the same settings as the old database, reloads it, and replaces the old database. However, you can change the initialization options as necessary by specifying other supported dbunload options (such as -ap to change the page size or -et to enable table encryption). If you use this option, there can be no other connections to the database, and the database connection must be local, not over a network. This option is not supported on Windows Mobile or when rebuilding version 9 or earlier databases on Mac OS X on Intel. If you specify an optional *directory*, the transaction log offsets are reset for replication purposes, and the transaction log from the old database is moved to the specified directory. The named directory should be the directory that holds the old transaction logs used by the Message Agent. The transaction log management is handled only if the database is used in replication: if there is no SQL Remote publisher, then the old transaction log is not needed and is deleted instead of being copied to the specified directory. See "Backing up databases involved in synchronization and replication" on page 918. When the new database is created, the dbspace file names have an R appended to the file name to prevent file name conflicts if the dbspace file for the new database is created in the same directory as the dbspace for the original database. For example, if an unloaded database has a dbspace called library in the file *library.db*, then the library dbspace for the new database is *library.dbR*. If you are rebuilding an encrypted database, the encryption key for the original and new databases must be the same. Using the -ar option resets the database truncation points to zero.
-c *"keyword=value; ..."*	Specifies the connection parameters for the source database. For a description of the connection parameters, see "Connection parameters" on page 265. The user ID should have DBA authority to ensure that the user has permissions on all the tables in the database. For example, the following statement unloads the sample database, connecting as user ID DBA with password sql. The data is unloaded into the *c:\unload* directory. `dbunload -c "DBF=samples-dir\demo.db;UID=DBA;PWD=sql" c:\unload` For information about *samples-dir*, see "Samples directory" on page 392.

Option	Description
-cm { **sql** | **dbinit** }	Displays in the server messages window the CREATE DATABASE or dbinit command to create a database that is the same as the one being unloaded. If -an is also specified, the command that is displayed is the command to create the new database. • **sql** Displays the CREATE DATABASE statement that is written to the *reload.sql* file. • **dbinit** Displays the Initialization utility (dbinit) command. When displaying the statement or command for an existing strongly-encrypted database (-an is not specified) the encryption key cannot be obtained from the database, so a question mark (?) appears in the ENCRYPTED clause or -ek option. The creation command or statement is not displayed if you unload a database that was created with a version 10 or earlier database server.
-cp	Compresses the table data output files by appending the COMPRESSED keyword to the UNLOAD TABLE statements it executes. This option has no effect when specified with -an or -ar.
-d	Does not generate any of the database definition commands (CREATE TABLE, CREATE INDEX, and so on); *reload.sql* contains statements to reload the data only.

-dc	Forces all computed columns in the database to be recalculated. By default, computed column values are not recalculated. When the -dc option is specified, a new section is added to the *reload.sql* script to recompute computed columns. Statements of the following form are added.
	`ALTER TABLE "owner"."table-name"`
	`ALTER "computed-column" SET COMPUTE (compute-expression);`
	If your tables contain context-sensitive computed values, such as CURRENT DATE, it is recommended that you use the ALTER TABLE statement to recalculate computed column values instead of using the -dc option. See “ALTER TABLE statement” [*SQL Anywhere Server - SQL Reference*].

-e *table*, ...	Excludes the specified tables from the *reload.sql* file. Table names are always case insensitive, even in case sensitive databases. A *reload.sql* file created with the -e option should not be used to rebuild a database because the file will not include all the database tables. If a table has foreign keys referring to it, the database cannot be rebuilt without the contents of the table. It is recommended that you only use the -e option with the -d option to unload data for all tables except those identified by -e.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-ea *algorithm*	Specifies the encryption algorithm used for database or table encryption (-et). Specify `-ea simple` for simple encryption (do not specify -ek or -ep). Simple encryption is equivalent to obfuscation and is intended only to keep data hidden in the event of casual direct access of the database file, to make it more difficult for someone to decipher the data in your database using a disk utility to look at the file. For greater security, specify AES or AES256 for 128-bit or 256-bit strong encryption, respectively. Specify AES_FIPS or AES256_FIPS for 128-bit or 256-bit FIPS-approved encryption, respectively. For strong encryption, you must also specify the -ek or -ep option. For more information about strong encryption, see "Strong encryption" on page 1131. To create a database that is not encrypted, specify `-ea none`, or do not include the -ea option (and do not specify -e, -et, -ep, or -ek). If you do not specify the -ea option, the default behavior is as follows: ● -ea none, if -ek, -ep, or -et is not specified ● -ea AES, if -ek or -ep is specified (with or without -et) ● -ea simple, if -et is used without -ek or -ep Algorithm names are case insensitive. **Separately licensed component required** ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations. See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].
-ek *key*	Specifies an encryption key in the dbunload command for the new database created if you unload and reload a database (using the -an option). If you create a strongly encrypted database, you must provide the encryption key to use the database or transaction log in any way. The algorithm used to encrypt the database is the algorithm specified by the -ea option. If you specify the -ek option without specifying -ea, the AES algorithm is used. See "Strong encryption" on page 1131. Protect your key. Be sure to store a copy of your key in a safe location. A lost key will result in a completely inaccessible database, from which there is no recovery.
-ep	Prompts for an encryption key for the new database created if you unload and reload your database using the -an option. It provides an extra measure of security by never allowing the encryption key to be seen in clear text. If you specify -ep without specifying -an, the -ep option is ignored. If you specify -ep and -an, you must input the encryption key twice to confirm that it was entered correctly. If the keys don't match, the unload fails. See "Strong encryption" on page 1131.

Option	Description

-er	Removes encryption from encrypted tables during an unload procedure.
	
	When rebuilding a database that has table encryption enabled, you must specify either -er or -et to indicate whether the new database has table encryption enabled, otherwise you get an error when attempting to load the data into the new database.
	
	The following command unloads a database (*mydata.db*) that has encrypted tables, into a new database (*mydatacopy.db*) that does not have table encryption enabled, removing encryption from any encrypted tables:
	
	`dbunload -an c:\mydatacopy.db -er`
	`-c "UID=DBA;PWD=sql;`
	`DBF=c:\mydata.db;`
	`DBKEY=29bN8cj1z"`

-et	Enables database table encryption in the new database (-an or -ar must also be specified). If you specify the -et option without the -ea option, the AES algorithm is used. If you specify the -et option, you must also specify -ep or -ek. You can change the table encryption settings for the new database to be different than those of the database you are unloading.
	
	When rebuilding a database that has table encryption enabled, you must specify either -er or -et to indicate whether the new database has table encryption enabled, otherwise you get an error when attempting to load the data into the new database.
	
	The following example unloads a database (*mydata.db*) that has tables encrypted with the simple encryption algorithm, into a new database (*mydatacopy.db*) that has table encryption enabled, and uses AES_FIPS encryption with the key 34jh:
	
	`dbunload -an c:\mydatacopy.db -et -ea AES_FIPS`
	`-ek 34jh`
	`-c "UID=DBA;PWD=sql;DBF=c:\mydata.db"`

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-g	**Materialized views** By default, materialized views defined as MANUAL REFRESH are not initialized after a reload. If you want these materialized views to be initialized as part of the reload process, specify the -g option. Specifying -g causes the database server to execute the sa_refresh_materialized_views system procedure. See "sa_refresh_materialized_views system procedure" [*SQL Anywhere Server - SQL Reference*]. When deciding whether to use the -g option, consider that initializing all materialized views may cause the reload process to take significantly longer to complete. On the other hand, not using the -g option means that the first query that attempts to use an uninitialized materialized view must wait while the database server initializes the view, which may cause an unexpected delay. If you do not use the -g option, you can also manually initialize materialized views after the reload completes. See "Initialize materialized views" [*SQL Anywhere Server - SQL Usage*]. **Text indexes** By default, text indexes defined as MANUAL REFRESH are not initialized after a reload. If you want the text indexes initialized as part of the reload process, specify the -g option. Specifying -g causes the database server to execute the sa_refresh_text_indexes system procedure. See "sa_refresh_text_indexes system procedure" [*SQL Anywhere Server - SQL Reference*].
-ii	Uses the UNLOAD statement to extract data from the database, and uses the LOAD statement in the *reload.sql* file to repopulate the database with data. This is the default.
-ix	Uses the UNLOAD statement to extract data from the database, and uses the Interactive SQL INPUT statement in the *reload.sql* file to repopulate the database with data.
-k	Populates the sa_diagnostic_auxiliary_catalog table. This table maps database object IDs for tables, users, procedures, and so on, from the source database to the tracing database. It also causes all histograms to be unloaded/reloaded. This option is used when creating a tracing database, that is, a database that receives diagnostic tracing information. The sa_diagnostic_auxiliary_catalog table allows the server to simulate conditions that were present when tracing data was captured (for example, for use with Index Consultant, or application profiling). This option is most useful when specified with the -n option. See "Advanced application profiling using diagnostic tracing" [*SQL Anywhere Server - SQL Usage*] and "sa_diagnostic_auxiliary_catalog table" [*SQL Anywhere Server - SQL Reference*].
-kd	Reloads the database into a single dbspace file. This option is useful for creating a tracing database if the computer where the tracing database is being created does not have the same directory structure as the production database. See "Creating an external tracing database" [*SQL Anywhere Server - SQL Usage*].

Option	Description
-l	Forces the current value of SYSTABCOL.max_identity to be preserved across a database rebuild. By default, when a database containing tables with autoincrement columns is rebuilt, the database server calculates the next available value for each autoincrement column based on the current contents of the tables. Usually this is enough; however, if rows have been deleted from the end of the range of values, values can be reused, which is not desirable. Specifying the -l option adds calls to the sa_reset_identity system procedure to the generated *reload.sql* script for each table that contains an autoincrement value, preserving the current value of SYSTABCOL.max_identity. See also: ● "Reloading tables with autoincrement columns" [*SQL Anywhere 12 - Changes and Upgrading*] ● "SYSTABCOL system view" [*SQL Anywhere Server - SQL Reference*] ● "sa_reset_identity system procedure" [*SQL Anywhere Server - SQL Reference*]
-m	Does not preserve user IDs for databases involved in replication.
-n	Does not unload database data; *reload.sql* contains SQL statements to build the structure of the database only. If you want the *reload.sql* file to contain LOAD TABLE or INPUT statements, use -nl instead.
-nl	Unloads the structure (the same behavior as the -n option), but the resulting *reload.sql* file also includes LOAD TABLE or INPUT statements for each table. No user data is unloaded when this option is used. When you specify -nl, you must also include a data directory so that the LOAD/INPUT statements can be generated, even though no files are written to the directory. This option allows you to generate a reload script without unloading data. You can unload the data by specifying -d. If a database contains a table whose data should not be unloaded, unloading the data for that table can be skipped using `dbunload -d -e table-name`.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-no	Unloads the database objects ordered by name. By default, dbunload generates objects in the order they were created. Specifying the -no option may be useful for comparing database schemas when the databases contain the same objects, but the creation order was different. Object definitions are grouped by object type in alphabetical order in the *reload.sql* file if -no is specified: • users • group memberships • tables • indexes and foreign keys • views • procedures • functions • triggers • events • web services The object definitions are output in owner,name order. Sometimes a third element such as a foreign key, role name, or trigger name is included in the ordering. The -no option cannot be used with the -n, -nl, -ar, -an, or -ac option. To simplify comparisons, it is recommended that you use -no option when comparing the reload scripts for databases that were created using the same version of the database server because of minor differences in the object definitions. **Caution** The generated file should not be used to create a new database because the object creation order can be important. For example, if procedure p2 calls procedure p1 and p1 returns a result set, it may be important to define p1 before p2. Attempting to execute a *reload.sql* script generated with -no option results in an error.
-o *filename*	Writes output messages to the named file. The location of this file is relative to dbunload.
-p *char*	Replaces the default escape character (\) for external unloads (dbunload -x option) with another character. This option is available only when you run this utility from a command prompt.
-q	Runs in quiet mode—messages are not displayed. This option is available only when you run this utility from a command prompt. If you specify -q, you must also specify -y or the unload will fail if *reload.sql* already exists.
-qc	Closes the messages window once the unload completes. By default, the dbunload messages window remains open until a user closes it. This option is only available on Windows Mobile.

Option	Description
-qr	Prevents progress messages from being created and displayed when loading tables and creating indexes.
-r *reload-file*	Modifies the name and directory of the generated reload command file. The default is *reload.sql* in the current directory. The directory is relative to the current directory of the client application, not the server.
-t *table,...*	Specifies a list of tables to be unloaded. By default, all tables are unloaded. Together with the -n option, this allows you to unload a set of table definitions only. Table names are always case insensitive, even in case sensitive databases. A *reload.sql* file created with the -t option should not be used to rebuild a database because the file will not include all the database tables. If a table has foreign keys referring to it, the database cannot be rebuilt without the contents of the table. It is recommended that you only use the -t option with the -d option to unload data for the tables identified by -t.
-u	Prevents an index from being used to order data. Use this option if you are unloading a database with a corrupt index, so that the corrupt index is not used to order the data. Normally, the data in each table is ordered by the primary key or clustered index if one is defined for the table.
-v	Displays the name of the table being unloaded, and the number of rows that have been unloaded. This option is available only when you run dbunload from a command prompt.
-xi	Performs an external unload by unloading data to the dbunload client, and then using the LOAD statement in the generated reload command file, *reload.sql*, to repopulate the database with data.
-xx	Performs an external unload by unloading data to the dbunload client, and then using the Interactive SQL INPUT statement in the generated reload command file, *reload.sql*, to repopulate the database with data.
-y	Replaces existing command files without prompting for confirmation. If you specify -q, you must also specify -y or the unload will fail if dbunload detects that a command file already exists. There are special considerations for unloading databases involved in replication. See "Extracting remote databases" [*SQL Remote*] and "Upgrading SQL Remote" [*SQL Anywhere 12 - Changes and Upgrading*].
directory	Specifies the directory where the unloaded data is to be placed. The *reload.sql* command file is always relative to the current directory of the user.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Remarks

Upgrading to version 12
For information about rebuilding an existing database into a version 12 database, see "Upgrading SQL Anywhere" [*SQL Anywhere 12 - Changes and Upgrading*].

When using dbunload with a version 10.0.0 or later database, the version of dbunload used must match the version of the database server used to access the database. If an older version of dbunload is used with a newer database server, or vice versa, an error is reported.

With the Unload utility, you can unload a database and put a set of data files in a named directory. The Unload utility creates an Interactive SQL command file to rebuild your database. It also unloads all the data in each of your tables into files in the specified directory, in comma-delimited format. Binary data is properly represented with escape sequences.

An internal unload/reload unloads information about the current status of each user by issuing UPDATE ISYSUSER statements. An external unload/reload does not include this information and the status of all users is reset. See "Managing login policies" on page 471.

When you rebuild a database by unloading and reloading it, the rebuilt database may be smaller than the original database. This decrease in database size may be the result of indexing changes in SQL Anywhere, and does not indicate a problem or a loss of data.

If you rebuild a version 11 or earlier database using the Unload utility (dbunload), the new database has global checksums enabled by default, even if the original database had checksums turned off. Although it is not recommended, you can disable global checksums for a database using the ALTER DATABASE statement. See "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*] and "Using checksums to detect corruption" on page 928.

Note
Version 9 and earlier databases that require recovery cannot be reloaded with version 10 or later of the Unload utility (dbunload). You must reload the database with version 9 or earlier of dbunload.

You can also use the Unload utility to directly create a new database from an existing one. This avoids potential security problems with the database contents being written to ordinary disk files.

If you only want to unload table data, you can do so in one step using the **Unload Data** window in Sybase Central.

For more information, see "Export data with the Unload Data window" [*SQL Anywhere Server - SQL Usage*].

There are special considerations for unloading databases involved in replication. See "Extracting remote databases" [*SQL Remote*].

You can access the Unload utility in the following ways:

- From Sybase Central, using the **Unload Database Wizard**. See "Export data with the Unload Database Wizard" [*SQL Anywhere Server - SQL Usage*].
- At a command prompt, using the dbunload command. This is useful for incorporation into batch or command files.

The Unload utility should be run by a user ID with DBA authority. This is the only way you can be sure of having the necessary privileges to unload all the data. In addition, the *reload.sql* file should be run by a user with DBA authority. (Usually, it is run on a new database where the only user ID is DBA with password sql.)

The database server -gl option controls the permissions required to unload data from the database. See "-gl dbeng12/dbsrv12 server option" on page 188.

The dbo user ID owns a set of system objects in a database, including views and stored procedures.

The Unload utility does not unload the objects that were created for the dbo user ID during database creation. Changes made to these objects, such as redefining a system procedure, are lost when the database is unloaded. Any objects that were created by the dbo user ID since the initialization of the database are unloaded by the Unload utility, and so these objects are preserved.

When you unload a database, changes to permissions on system objects are not unloaded. You must grant or revoke these permissions in the new database.

Tip
Before rebuilding your database, it is recommended that you validate the reload process by reloading the database without any data, by running a command similar to the following:

```
dbunload -n -an new.db -c "UID=your-user-id;PWD=your-password;DBF=original-database-file"
```

You should fix any problems that are identified in the original database before rebuilding it.

In the default mode, or if -ii or -ix is used, the directory used by dbunload to hold the data is relative to the database server, not to the current directory of the user.

If -xi or -xx is used, the directory is relative to the current directory of the user.

For more information about supplying a file name and path in this mode, see "UNLOAD statement" [*SQL Anywhere Server - SQL Reference*].

If no list of tables is supplied, the whole database is unloaded. If a list of tables is supplied, only those tables are unloaded.

Unloaded data includes the column list for the LOAD TABLE statements generated in the *reload.sql* file. Unloading the column list facilitates reordering of the columns in a table. Tables can be dropped or recreated, and then repopulated using *reload.sql*.

The LOAD TABLE statements generated by dbunload turn off check constraints and computed columns.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

Databases with materialized views
It is recommended that you refresh the materialized views in your database after rebuilding the database. See "Refresh manual views" [*SQL Anywhere Server - SQL Usage*].

Databases running diagnostic tracing
Tracing information is not unloaded as part of a database unload or reload operation. If you want to transfer tracing information from one database to another, you must do so manually by copying the contents of the sa_diagnostic_* tables; however, this is not recommended.

Internal versus external unloads and reloads

The following options offer combinations of internal and external unloads and reloads: -ii, -ix, -xi, and -xx. A significant performance gain can be realized using internal commands (UNLOAD/LOAD) versus external commands (Interactive SQL INPUT and OUTPUT statements). However, internal commands are executed by the server so that file and directory paths are relative to the location of the database server. Using external commands, file and directory paths are relative to the current directory of the user.

In Sybase Central, you can specify whether to unload relative to the server or client. See "UNLOAD statement" [*SQL Anywhere Server - SQL Reference*].

When you use an external unload and reload to unload, reload, or rebuild a database and the character set of the database is incompatible with the character set of the host system on which dbunload is running, character set conversion may cause data to be corrupted as it is converted between the database character set and the host system's character set.

To avoid this problem, specify the database character set in the connection string for the database (-c and -ac options). For example, if the database character set is UTF-8, you should include "charset=utf-8" in the connection strings:

```
dbunload -c UID=user-ID;PWD=password;
CHARSET=utf-8;DBF=filename -ac UID=user-ID;
PWD=password;CHARSET=utf-8;Host=host-name -xx
```

When you perform an external unload, the beginning of the *reload.sql* includes a commented CREATE DATABASE statement. This statement can be used to create a database that is equivalent to the one being unloaded.

If the unloaded database was created with version 9 or earlier of SQL Anywhere and had a custom collation, the COLLATION clause appears as follows:

```
COLLATION collation-label DEFINITION collation-definition
```

where *collation-definition* is a string that specifies the custom collation.

The only way to preserve a custom collation is to rebuild the database in a single step (internal unload). If you choose to unload the database, and then load the schema and data into a database that you create, then you must use one of the supplied collations.

If the unloaded database was created with strong encryption, the value of the KEY clause in the CREATE DATABASE statement appears as three question marks (**???**).

Failed unloads

If a failure occurs during an internal rebuild of a database using -ar or -an, after the table data has been reloaded and any indexes on the table have been rebuilt, dbunload creates a file named *unprocessed.sql* in the current directory. This file contains all the statements that were not executed as a result of the failure, and also includes the statement that caused the failure as a comment. The following is an example of an *unprocessed.sql* file:

```
-- The database reload failed with the following error:
-- ***** SQL error: the-SQL-ERROR
-- This script contains the statements that were not executed as a
-- result of the failure. The statement that caused the failure is
-- commented out below. To complete the reload, correct the failing
-- statement, remove the surrounding comments and execute this script.
/*
the failing statement
go

*/

setuser "DBA"
go

... the remainder of the statements to be processed
```

Having this file gives you the opportunity to correct, remove, or alter the failing statement(s). The *unprocessed.sql* file is only created after all the table data and referential integrity constraints have been reloaded. Using Interactive SQL, you can connect to the new database and execute the updated *unprocessed.sql* file. This allows you to complete the rebuild of the database without having to start the rebuild over again, which can save considerable time.

When the *unprocessed.sql* file is generated, dbunload stops and returns a failed error code to make other tools or scripts aware of the failed rebuild.

If a failure occurs during an internal rebuild of a database using -ar, the new database and transaction log file have the *.db**r*** and *.log**r*** file extensions, respectively. Use the following steps to apply the unprocessed.sql file and finish the reload manually:

1. Start the new database.
2. Apply the updated *unprocessed.sql* file.
3. Shut down the database.
4. Move *original-name.db* and *original-name.log* to a new directory.
5. Rename the *original-name.db**r*** and *original-name.log**r*** files to *original-name.db* and *original-name.log* respectively.
6. Run the following command:

   ```
   dblog -t original-name.log original-name.db
   ```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Encrypted databases

When you rebuild a database that has table encryption enabled, you must specify either -er or -et to indicate whether the new database has table encryption enabled, otherwise you get an error when attempting to load the data into the new database.

If you want to unload a strongly encrypted database, you must provide the encryption key. You can use the DatabaseKey (DBKEY) connection parameter to provide the key in the command. Alternatively, if you want to be prompted for the encryption key rather than entering it in plain view, you can use the -ep server option as follows:

```
dbunload -c "DBF=myenc.db;START=dbeng12 -ep"
```

If you are using the -an option to unload a database and reload into a new one, and you want to use the -ek or -ep options to set the encryption key for the new database, keep the following in mind:

- If the original database is strongly encrypted, you need to specify the key for the original database using the DatabaseKey (DBKEY) connection parameter in the -c option, rather than using -ek or -ep.
- Using the -ek and -ep options, it is possible to unload an unencrypted database and reload into a new, strongly encrypted database. When you use -ep and -an, you must confirm the key correctly or the unload fails.
- If the original database is strongly encrypted, but the -ek and -ep options are not used, then the new database will be encrypted with simple encryption.
- The -ek and -ep options are ignored if -an is not specified. The dbunload -ek and -ep options apply to a new database, while the database server (dbeng12/dbsrv12) options and DBKEY= apply to existing databases.
- When rebuilding databases involved in synchronization or replication, dbunload assumes that the encryption key specified with the -ek or -ep option is the encryption key of the original database, and the encryption key of the newly-rebuilt database.

For more information about encryption, see "-ep dbeng12/dbsrv12 server option" on page 179 and "DatabaseKey (DBKEY) connection parameter" on page 281.

Rebuilding a database

To unload a database, first ensure that the database is not already running. Then, run dbunload, specifying a DBA user and password, and referencing the database with the DBF= connection parameter.

To reload a database, create a new database and then run the generated *reload.sql* command file through Interactive SQL.

To combine the unload and reload steps, follow the directions for unloading above, but add the -an option to specify the name of the new database file. See the descriptions of the -ac and -an options.

Upgrade utility (dbupgrad)

Note
You cannot use the Upgrade utility (dbupgrad) to upgrade SQL Anywhere 9.0.2 and earlier databases to SQL Anywhere 12. To upgrade SQL Anywhere 9.0.2 and earlier databases to SQL Anywhere 12, you must rebuild the database by performing an unload and reload. See "Upgrading SQL Anywhere" [*SQL Anywhere 12 - Changes and Upgrading*].

Updates the system tables and views, adds new database options, and recreates all system stored procedures. Installs jConnect support and changing support for Java in the database.

Syntax

dbupgrad [*options*]

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-c *"keyword=value; ..."*	Specifies connection parameters. See "Connection parameters" on page 265. The user ID must have DBA authority. For example, the following command upgrades a database called sample12 and does not install jConnect support, connecting as user DBA with password sql: `dbupgrad -c "UID=DBA;PWD=sql;DBF=c:\sa12\sample12.db" -i`
-i	Excludes jConnect metadata support. If you want to use the jConnect JDBC driver to access system catalog information, you should not use this option. You can still use jConnect when this option is specified, as long as you do not access system catalog information. If you want, you can add jConnect metadata support later using Sybase Central or the ALTER DATABASE UPGRADE statement. See "Installing jConnect system objects into a database" [*SQL Anywhere Server - Programming*] and "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*].
-o *filename*	Writes output messages to the specified file.
-q	Runs in quiet mode—messages and windows are not displayed.

Remarks

Caution
You should always back up your database files before upgrading. If you apply the upgrade to the existing files, then these files become unusable if the upgrade fails. For information about backing up your database, see "Backup and data recovery" on page 887.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The dbupgrad utility upgrades a database created with earlier versions of the software to enable features from the current version of the software. The earliest version that can be upgraded is SQL Anywhere 10.0.0. While later versions of the database server can run against databases that were created with earlier releases of the software, some of the features introduced since the version that created the database are unavailable unless the database is upgraded.

An error message is returned if you use the Upgrade utility to upgrade a database that is currently being mirrored.

Databases with materialized views
It is recommended that you refresh the materialized views in your database after upgrading the database. See "Refresh manual views" [*SQL Anywhere Server - SQL Usage*].

You can use the Upgrade utility to update the system tables and views, add new database options, restore database options, and recreate all system stored procedures, and install jConnect support and change support for Java in the database.

Not all features made available
Features that require a physical reorganization of the database file are not made available by dbupgrad. Such features include index enhancements and changes in data storage. To obtain the benefits of these enhancements, you must unload and reload your database. See "Upgrading SQL Anywhere" [*SQL Anywhere 12 - Changes and Upgrading*].

As new versions and software updates become available for SQL Anywhere, you can use the Upgrade utility to take advantage of the new features.

Upgrading a database does not require you to unload and reload your database.

If you want to use replication on an upgraded database, you must also archive your transaction log and start a new one on the upgraded database.

You can access the Upgrade utility in the following ways:

- From Sybase Central, using the **Upgrade Database Wizard**.
- From Interactive SQL, using the ALTER DATABASE UPGRADE statement. See "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*].
- At a command prompt, using the dbupgrad command.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

Validation utility (dbvalid)

Validates the indexes and keys on some or all the tables and materialized views in a database.

Syntax

dbvalid [*options*] [*object-name*, ...]

Option	Description
@data	Reads in options from the specified environment variable or configuration file. See "Using configuration files" on page 763. If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See "File Hiding utility (dbfhide)" on page 794.
-c *"keyword=value; ..."*	Specifies database connection parameters. For a description of the connection parameters, see "Connection parameters" on page 265. The user ID must have DBA authority or VALIDATE authority. For example, the following command validates the database, including all tables and materialized views for *c:\salesdata.db*, connecting as user DBA with password sql: `dbvalid -c "UID=DBA;PWD=sql;DBF=c:\salesdata.db"`
-d	Validates that all table pages in the database belong to the correct object, and performs a checksum validation. The -d option does not include validation of data or indexes. The -d option cannot be used with the -i, -s, or -t options When you specify -d, dbvalid reads each page in the database and validates the checksum on any page that has a checksum, regardless of whether a database was created with global checksums, has global checksums disabled, or has used write checksums. Dbvalid also validates checksums for metadata pages that by default have checksums, such as the database's definition page. See "Using checksums to detect corruption" on page 928.
-fx	Validates every row of the table, and make sure that the number of rows in the table matches the number of rows in each index associated with the table. This option does not perform individual index lookups for each row. Using this option can significantly improve performance when validating large databases with a small cache.
-i	Validates the specified index.
-o *filename*	Writes output messages to the named file.
-q	Does not display output messages to the client. You can still log the messages to file using the -o option, however.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Option	Description
-s	Validates the database using checksums. Checksums are used to determine whether a database page has been modified on disk. Checksum validation reads each page of the database from disk and calculates its checksum if the page has a checksum. If the calculated checksum is different from the checksum stored on the page, the page has been modified on disk and an error is returned. The page numbers of any invalid pages appear in the database server messages window. The -s option cannot be used in conjunction with -d, -i, -t, or either of the -f options.
-t	Specifies a list of *object-name* values, which represents a list of tables and materialized views.
object-name	Specifies the name of the table or materialized view to validate. If -i is used, *object-name* refers to an index to validate instead.

Remarks

By default, dbvalid validates all the tables, materialized views, and indexes, in the database, and validates the database itself. If you start database validation while the database cleaner is running, the validation does not run until the database cleaner is finished running. See "sa_clean_database system procedure" [*SQL Anywhere Server - SQL Reference*].

With the Validation utility, you can validate the indexes and keys on some of, or all, the tables and materialized views in a database. You can also use the Validation utility to verify that all table pages in the database belong to the correct object, and that page checksums are correct. By default, dbvalid validates all the tables and materialized views in the database (the same behavior as the -t option).

For each table or materialized view, the Validation utility scans the entire object, and then looks up each record in every index and key defined on the table. You can also use the Validation utility to verify that all table pages in the database belong to the correct object, and that page checksums are correct. To run the Validation utility, you must have either DBA or VALIDATE authority.

You can also access the Validation utility in the following ways:

- From Sybase Central, using the **Validate Database Wizard**. See "Validate a database" on page 930.
- From Interactive SQL, using the VALIDATE statement. See "VALIDATE statement" [*SQL Anywhere Server - SQL Reference*].

The Validation utility can be used in combination with regular backups to give you confidence in the integrity of the data in your database. If you want to validate the backup copy of your database, it is recommended that you make a copy of the backup and validate the copy. Doing this ensures that you do not make changes to the file that is used in recovery. See "Backup and data recovery" on page 887.

Caution
Backup copies of the database and transaction log must not be changed in any way. If there were no transactions in progress during the backup, or if you specified BACKUP DATABASE WITH CHECKPOINT LOG RECOVER or WITH CHECKPOINT LOG NO COPY, you can check the validity of the backup database using read-only mode or by validating a copy of the backup database.

However, if transactions were in progress, or if you specified BACKUP DATABASE WITH CHECKPOINT LOG COPY, the database server must perform recovery on the database when you start it. Recovery modifies the backup copy, which prevents subsequent transaction log files from the original database from being applied.

If running the Validation utility starts a database automatically, the database starts in read-only mode. This prevents changes from being made to the database if the validation is part of a backup or recovery plan.

If the Validation utility connects to a running database that was not started in read-only mode, the utility displays a warning. This warning is a reminder that the database being validated cannot be used as part of a recovery plan. Because of the way backups are performed, most databases created by dbbackup are marked as needing recovery. If the database you are validating requires recovery and you want to force it to start as read-write, you can either start the database before running dbvalid or specify a valid value for the DBS connection parameter. See "DatabaseSwitches (DBS) connection parameter" on page 283.

Both of the following commands allow dbvalid to run if the *mycopy.db* database needs to be recovered:

```
dbvalid -c "UID=DBA;PWD=sql;DBF=mycopy.db;DBS=-n mycopy"
dbvalid -c "UID=DBA;PWD=sql;DBF=mycopy.db;DBS=-dh"
```

Caution
Validating a table or an entire database should be performed while no connections are making changes to the database; otherwise, errors may be reported indicating some form of database corruption even though no corruption actually exists.

The Validation utility may return warnings about checksum violations for databases that do not have global checksums enabled. This is because the database server automatically still calculates checksums for critical database pages, regardless of whether checksums are enabled. A database may also have checksums on some pages because it was started with write checksums. See "Using checksums to detect corruption" on page 928.

The database server also creates checksums automatically for databases running on Windows Mobile and for databases running on storage media that may be less reliable, such as removable drives. See "Using checksums to detect corruption" on page 928.

Validation requires exclusive access to each table. For this reason, it is best to validate when there is no other activity on the database.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

For more information about specific checks made during validation, see "VALIDATE statement" [*SQL Anywhere Server - SQL Reference*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Version Diagnostic utility (dbversion)

Returns information about the specified executable.

Syntax

dbversion *executable-name*

Remarks

This utility is only available on Unix, and returns information about SQL Anywhere executables.

See also

- "-v dbeng12/dbsrv12 server option" on page 234

Example

The following command:

```
$ dbversion /opt/sqlanywhere12/bin32/dbversion
```

returns information about the dbversion executable:

```
SQL Anywhere Version Diagnostic Utility Version 12.0.0.2413
/opt/sqlanywhere12/bin32/dbversion: dbversion xx 12 0 0 2413 linux 2008/04/02
23:31:54
nothr 32 production
```

Field	Description
dbversion	Returns the executable name.
xx	Returns a two-letter code designating an install type.
12	Returns the major version number.
0	Returns the minor version number.
0	Returns the patch number.
1238	Returns the build number.
linux	Returns the operating system code.
2008/04/02 23:31:54	Returns the build time/datestamp.
nothr	Returns the threading model (nothr or posix).
32	Returns the bitness of the executable (32 or 64).
production	Returns either production or debug.

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Maintaining your database

This section describes how to back up database files and how to use events and schedules to automate database administration.

Backup and data recovery

A **backup** is a full or partial copy of the information in a database, held in a physically separate location. If the database becomes unavailable, you can **restore** it from the backup. You can use your backups to restore all committed changes to the database up to the time it became unavailable.

Backing up a running database provides a snapshot of the database where the data is in a consistent state, even though other users are modifying the database.

If the operating system or database server fails, or the database server does not shut down properly, then the database must be recovered. On database startup, the database server checks whether the database was shut down cleanly at the end of the previous session. If it was not, the database server executes an automatic recovery process to restore all changes up to the most recently committed transaction.

SQL Anywhere tools make **online** backups that are executed against a running database. You must have BACKUP authority or REMOTE DBA authority to make online backups of a database. You can make **offline** backups by copying the database files when the database is not running.

See also

- "Types of backup" on page 888
- "Backup quick start" on page 887
- "Backup utility (dbbackup)" on page 767
- "BACKUP statement" [*SQL Anywhere Server - SQL Reference*]
- "Designing a backup and recovery plan" on page 913
- "Recovering your database" on page 902

Backup quick start

When you make a backup, you must decide where you want to store the backup files: on the database server computer or on the client computer. You must have BACKUP authority or REMOTE DBA authority to back up the database using the following procedures.

To make a server-side backup

- Run a BACKUP DATABASE statement. For example:

  ```
  BACKUP DATABASE DIRECTORY 'd:\\temp\\backup';
  ```

  This statement creates a backup copy of the database files in the directory *d:\temp\backup* on the server computer.

Alternatively, you can run dbbackup with the -s option to create the backup. For example:

```
dbbackup -s -c
"Host=sample_host;SERVER=myserver;DBN=demo;UID=DBA;PWD=sql"
"c:\SQLAnybackup"
```

To make a client-side backup

- Run the Backup utility (dbbackup) on the client computer. For example:

```
dbbackup -c "Host=sample_host;SERVER=myserver;DBN=demo;UID=DBA;PWD=sql"
"c:\SQLAnybackup"
```

See also

- “Backup utility (dbbackup)” on page 767
- “BACKUP statement” [*SQL Anywhere Server - SQL Reference*]
- “Types of backup” on page 888
- “Designing a backup and recovery plan” on page 913
- “Recovering your database” on page 902

Types of backup

The following table summarizes the types of backup supported by SQL Anywhere:

Backup type	Description	More information
Online	A backup that is performed while the database is running.	See “Online and offline backups” on page 889.
Offline	A backup that is performed when the database is not running. This type of backup should only be performed when the database server has shut down properly.	See “Online and offline backups” on page 889.
Full	A full backup is a backup of the database files and the transaction log. Typically, full backups are interspersed with several incremental backups.	“Full backups” on page 889
Incremental	A backup of the transaction log only.	“Incremental backups” on page 890
Live	A continuous backup of the database that runs while the database is running.	“Live backups” on page 891

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Backup type	Description	More information
Archive	A collection of one or more files that together contain all the required information for the backup, including the main database file, the transaction log, and any additional dbspaces.	"Archive backups" on page 892
Image	A copy of the database file and/or the transaction log, each as separate files.	"Image backups" on page 893
Server-side	A backup made on the database server computer.	"Making a server-side backup" on page 894
Client-side	A backup made on the client computer.	"Making a client-side backup" on page 899

Online and offline backups

An offline backup is a copy of the database files. You should only perform an offline backup when the database is not running, and when the database server has shut down properly.

All of the tools included with SQL Anywhere, such as the Backup Database utility (dbbackup), BACKUP DATABASE statement, and Sybase Central wizards, perform online backups while the database is running.

Backing up a running database provides a snapshot of the database where the data is in a consistent state, even though other users are modifying the database.

See also

- "Types of backup" on page 888

Full backups

A **full backup** is a backup of both the database file and the transaction log. You must have BACKUP or REMOTE DBA authority to perform a full backup.

To make a full backup (overview)

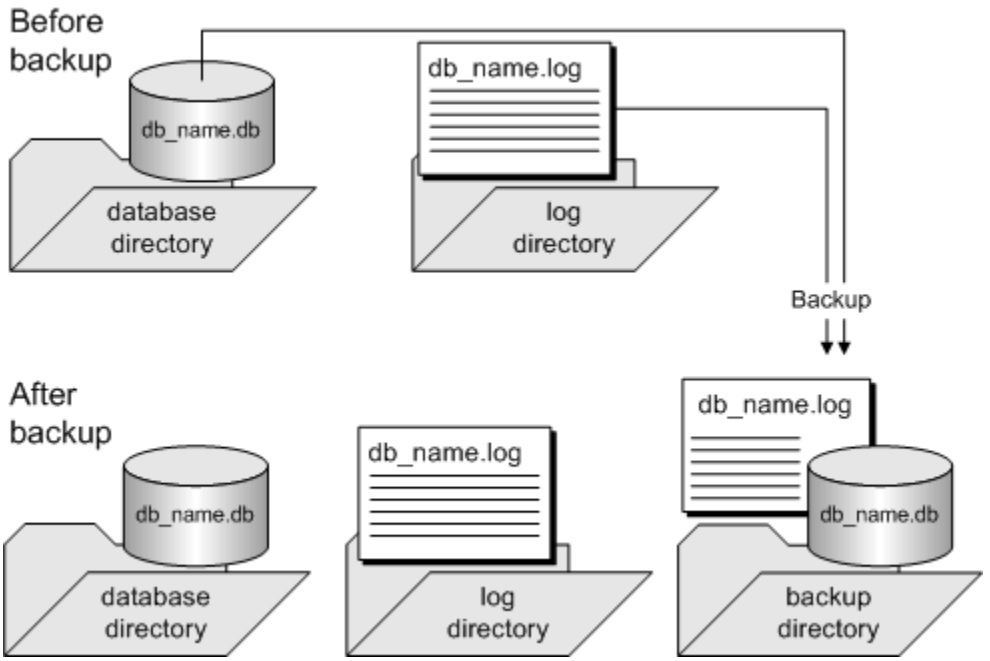
1. Perform a validity check on your database to ensure that it is not corrupt. You can use the Validation utility or the sa_validate stored procedure. See "Validate a database" on page 930.

2. Make a backup of your database file and transaction log.

For information about how to perform the backup operation, see:

- "Making a server-side backup" on page 894
- "Making a client-side backup" on page 899
- "Make a backup and delete the original transaction log" on page 921
- "Make a backup and rename the original transaction log" on page 919

The simplest form of backup is an image backup (which consists of a copy of the database file and/or the transaction log, each as separate files) that makes copies of the database file and transaction log, and leaves the transaction log in place without truncating or replacing it. All backups leave the database file in place. A full backup of this kind is illustrated in the following figure.



See also

- "Incremental backups" on page 890
- "Types of backup" on page 888

Incremental backups

An **incremental backup** is a backup of the transaction log only. Typically, full backups are interspersed with several incremental backups. See "Full backups" on page 889.

The backup copies of the database file and transaction log file have the same names as the online versions of these files. For example, if you make a backup of the sample database, the backup copies are called *demo.db* and *demo.log*. When you repeat the backup statement, choose a new backup directory to avoid overwriting the backup copies.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For more information about making a repeatable incremental backup command by renaming the backup copy of the transaction log, see "Rename the backup copy of the transaction log during backup" on page 920.

To make an incremental backup (overview)

1. Ensure that you have BACKUP or REMOTE DBA authority on the database.

2. Make a backup of your transaction log, not your database file.

See also

- "Making a server-side backup" on page 894
- "Making a client-side backup" on page 899
- "Make a backup and delete the original transaction log" on page 921
- "Make a backup and rename the original transaction log" on page 919
- "Full backups" on page 889
- "Types of backup" on page 888

Live backups

A **live backup** is a continuous backup of the database that helps protect against total computer failure. You can use the redundant copy of the transaction log to restart your system on a secondary computer.

If your system fails, the backed up transaction log can be used for a rapid restart of the system. However, depending on the load that the database server is processing, the live backup may lag behind and may not contain all committed transactions.

An alternative to a live backup is to use database mirroring. See "Introduction to database mirroring" on page 945.

You should normally run the dbbackup utility from the secondary computer.

If the primary computer becomes unusable, you can restart your database using the secondary computer. The database file and the transaction log hold the information needed to restart the database.

Live backups and regular backups

The live backup of the transaction log is always the same length or shorter than the active transaction log. When a live backup is running, and another backup restarts the transaction log (dbbackup -r or dbbackup -x), the live backup automatically truncates the live backup log and restarts the live backup at the beginning of the new transaction log.

See also

- "Make a live backup" on page 900
- "Restart from a live backup" on page 906
- "Types of backup" on page 888

Differences between live backups and transaction log mirrors

Both a live backup and a transaction log mirror provide a secondary copy of the transaction log. However, there are several differences between using a live backup and using a transaction log mirror:

- **In general, a live backup is made to a different computer** By running the Backup utility on a separate computer, the database server does not do the writing of the backed up log file, and the data transfer is done by the SQL Anywhere client/server communications system. Therefore, the performance impact is decreased and reliability is greater.

  Running a transaction log mirror on a separate computer is not recommended. It can lead to performance and data corruption problems, and stops the database server if the connection between the computers fails.

- **A live backup provides protection against a computer becoming unusable** Even if a transaction log mirror is kept on a separate device, it does not provide immediate recovery if the whole computer becomes unusable. You could consider an arrangement where two computers share access to a set of disks.

- **A live backup may lag behind the database server** A transaction log mirror contains all the information required for complete recovery of committed transactions. Depending on the load that the database server is processing, the live backup may lag behind the transaction log mirror and may not contain all the committed transactions.

Choosing a backup format

An **archive backup** copies the database file and the transaction log into one or more files, typically on a tape drive. An **image backup** makes a copy of the database files and/or the transaction log, each as separate files. You can only perform archive backups as server-side backups.

You should use an archive backup if you are backing up directly to tape. Otherwise, an image backup should be used because image backups are easier to restore.

Archive backups

An archive backup is a collection of one or more files that together contain all the required information for the backup, including the main database file, the transaction log, and any additional dbspaces. You can only perform archive backups as server-side backups. You can save an archive backup to either a file or a tape drive. Archive backups can be made using the BACKUP DATABASE statement or the **Backup Database Wizard** in Sybase Central.

When making archive backups, an extension is added to the file name you specify in the BACKUP statement for each file that is created (*.1*, *.2*, *.3*, and so on).

By default, archive backups skip some free pages, which can result in smaller and potentially faster backups. Free page elimination has no affect on the back up of transaction log files because transaction

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

log files do not contain free pages. So, databases with large transaction log files may not benefit as much from free page elimination as databases with small transaction log files.

You restore a database from an archive backup using the **Restore Database Wizard** in Sybase Central or using the RESTORE DATABASE statement.

Archive backups are supported on Windows and Unix platforms only. On Windows Mobile, only image backups are permitted. See "Image backups" on page 893.

For information about making an archive backup, see:

- "Use the Backup Database Wizard" on page 897
- "Use the BACKUP DATABASE statement to make a server-side backup" on page 894
- "Types of backup" on page 888

Image backups

An image backup consists of a copy of the database file and/or the transaction log, each as separate files.

You can make an image backup using the Backup utility (dbbackup), the **Create Backup Images Wizard**, or the BACKUP DATABASE statement. Image backups are available on all supported platforms, and are the only supported type of backup on Windows Mobile.

If you want to make a backup to tape, use an archive backup. See "Archive backups" on page 892.

See also

- "BACKUP statement" [*SQL Anywhere Server - SQL Reference*]
- "Backup utility (dbbackup)" on page 767
- "Use the BACKUP DATABASE statement to make a server-side backup" on page 894
- "Use the Create Backup Images Wizard" on page 897
- "Types of backup" on page 888

Backup and recovery restrictions

The database server prevents the following operations from being executed while a backup is in progress:

- Another backup, with the exception of a live backup.
- A checkpoint, other than one issued by the backup instruction.
- Any statement that causes a checkpoint. This includes data definition statements and the LOAD TABLE and TRUNCATE TABLE statements.

During recovery, including recovering backups, no action is permitted by other users of the database.

See also

- "Understanding the checkpoint log" on page 25

Making a server-side backup

Making a backup on the database server computer is generally faster than a backup on a client computer because the data does not have to be transported across the client/server communications system. To build a server-side backup into your application, use a SQL statement. The following methods are supported for making a server-side backup:

Tool	More information
BACKUP statement	"Use the BACKUP DATABASE statement to make a server-side backup" on page 894
Backup utility (dbbackup)	"Use the Backup utility (dbbackup) to make a server-side backup" on page 896
Backup Database Wizard	"Using Sybase Central to make a server-side backup" on page 896
Create Backup Images Wizard	"Using Sybase Central to make a server-side backup" on page 896
Create Maintenance Plan Wizard	"Using Sybase Central to make a server-side backup" on page 896
DBBackup function	"a_backup_db structure" [*SQL Anywhere Server - Programming*]
SQL Anywhere Volume Shadow Copy Service (dbvss)	"Using the SQL Anywhere Volume Shadow Copy Service (VSS)" on page 898

Both the Backup utility (dbbackup) and BACKUP DATABASE statement use physical device-level parallelism to decrease the time required to complete a backup operation. Parallel backups are not supported on Windows Mobile. See "Understanding parallel database backups" on page 926.

See also

- "Making a client-side backup" on page 899
- "Types of backup" on page 888

Use the BACKUP DATABASE statement to make a server-side backup

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

This topic describes a backup that leaves the transaction log untouched. For information about other transaction log management options when making a backup, see "BACKUP statement" [*SQL Anywhere Server - SQL Reference*].

The BACKUP statement makes an entry in the text file *backup.syb*. For information about the location of the *backup.syb* file, see "SALOGDIR environment variable" on page 384.

Make an image backup

To make an image backup (SQL)

- Use the following clauses to execute a backup statement:

```
BACKUP DATABASE
DIRECTORY directory-name;
```

For information about recovering from an image backup, see "Restore from an image backup" on page 904.

Make an archive backup

To make an archive backup to tape (SQL)

- Use the BACKUP statement with the following clauses:

```
BACKUP DATABASE
TO archive-root
[ ATTENDED { ON | OFF } ]
[ WITH COMMENT comment-string ];
```

If you set the ATTENDED option to OFF, the backup fails if it runs out of tape or disk space. If ATTENDED is set to ON, you are prompted to take an action when there is no more space on the backup archive device.

For information about recovering from an archive backup, see "Restore from an archive backup" on page 904.

Examples

The following statement makes an image backup of the current database and the transaction log, saves them to different files, and renames the existing transaction log.

```
BACKUP DATABASE
DIRECTORY 'd:\\temp\\backup'
TRANSACTION LOG RENAME;
```

The following statement makes an archive backup to the first tape drive on a Windows computer:

```
BACKUP DATABASE
TO '\\\\.\\tape0'
ATTENDED OFF
WITH COMMENT 'May 6 backup';
```

The first tape drive on Windows is `\\.\tape0` . Because the backslash is an escape character in SQL strings, each backslash is preceded by another.

See also

- "BACKUP statement" [*SQL Anywhere Server - SQL Reference*]
- "Archive backups" on page 892
- "Image backups" on page 893
- "Recovering your database" on page 902

Use the Backup utility (dbbackup) to make a server-side backup

This topic describes a backup that leaves the transaction log untouched. For information about other transaction log management options when making a backup, see "Backup utility (dbbackup)" on page 767.

The dbbackup utility makes an image backup that consists of a copy of the database file and/or the transaction log, each as separate files.

To make a backup, continuing to use the original transaction log (command line)

- If you are using the dbbackup utility, use the following syntax:

  ```
  dbbackup -c "connection-string" [ -t ] backup-directory
  ```

  Include the -t option only if you are making an incremental backup. See "Incremental backups" on page 890.

Example

The following example makes a backup on the database server computer in the directory *c:\SQLAnybackup*.

```
dbbackup -s -c "HOST=myhost;DBN=demo;UID=DBA;PWD=sql" "c:\SQLAnybackup"
```

See also

- "Image backups" on page 893
- "Restore from an image backup" on page 904
- "Recovering your database" on page 902

Using Sybase Central to make a server-side backup

To make a server-side backup from Sybase Central, use one of the following wizards:

- **Backup Database Wizard** This wizard creates an archive backup. You can specify a file name or tape drive where the backup is stored. See "Use the Backup Database Wizard" on page 897.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Create Backup Images Wizard** This wizard creates a copy of each database file while the database is running. To recover, you copy all the files back in place on the database server computer. See "Use the Create Backup Images Wizard" on page 897.

- **Create Maintenance Plan Wizard** This wizard lets you create a schedule for a variety of tasks, including backing up the database. You can choose to create an archive, a file image, or an incremental backup. See "Creating a maintenance plan" on page 915.

See also

- "Archive backups" on page 892
- "Image backups" on page 893
- "Incremental backups" on page 890

Use the Backup Database Wizard

The **Backup Database Wizard** creates an archive backup. When you make an archive backup in Sybase Central, you have the option of backing up the database directly to tape or to disk.

To make a backup (Sybase Central Backup Database Wizard)

1. Connect to the database as a user with BACKUP or REMOTE DBA authority.
2. Right-click the database and choose **Backup Database**.
3. Follow the instructions in the wizard.

See also

- "Archive backups" on page 892
- "Restore from an archive backup" on page 904
- "Recovering your database" on page 902

Use the Create Backup Images Wizard

The **Create Backup Images Wizard** creates a copy of each database file. To recover, copy all the files back in place on the database server computer.

This procedure describes the simplest kind of backup, which leaves the transaction log untouched and allows you to continue using the existing transaction log.

To make a backup (Sybase Central Create Backup Images Wizard)

1. Connect to the database as a user with BACKUP or REMOTE DBA authority.
2. Right-click the database and choose **Create Backup Images**.
3. Click **Next**.

4. In the **Which Database Do You Want To Back Up** list, select the database and click **Next**.
5. In the **Save The Backup Images In The Following Directory** field, type the name of a directory to save the backup copies.
6. Select an option in the **Which Files Do You Want To Back Up** list and click **Next**.
7. In the **What Do You Want To Do With The Transaction Log** list, click **Continue To Use The Same Transaction Log**.
8. Click **Next**.
9. Click **Finish**.
10. Click **Close**.

Tip
You can also access the **Create Backup Images Database Wizard** from Sybase Central by using the following methods:

- Selecting a database, and choosing **File** » **Create Backup Images**.
- Choose **Tools** » **SQL Anywhere 12** » **Create Backup Images**.

See also

- "Image backups" on page 893
- "Restore from an image backup" on page 904
- "Recovering your database" on page 902

Using the SQL Anywhere Volume Shadow Copy Service (VSS)

SQL Anywhere is compatible with the Microsoft Volume Shadow Copy Service (VSS). You can use VSS to create point-in-time snapshots of entire disk volumes or volume sets and to make copies of files that are open for exclusive use by applications such as the SQL Anywhere database server. VSS is supported on 32-bit Windows XP operating systems and on 32-bit and 64-bit editions of Windows 2003 and later operating systems, including Windows Vista.

By default, all SQL Anywhere databases can use the VSS service for backups if the SQL Anywhere VSS writer (*dbvss12.exe*) is running. You can use VSS without the SQL Anywhere VSS writer to back up databases. However, you might need to use the full SQL Anywhere recovery procedures to restore those databases. To prevent a database server from participating in the VSS service, include **-vss-** when starting the database server. Alternatively, you can use the Service utility (dbsvc) for Windows to specify when the VSS service is started.

How VSS works with SQL Anywhere:

- Your backup application sends a command to VSS to take a snapshot.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- VSS issues an **identify** command to the SQL Anywhere VSS writer (*dbvss12.exe*).
- VSS issues a **prepare to snapshot** command to suspend all transactions and write all modified pages to disk on all databases on all database servers. If transactions are not suspended on a database within 10 seconds, the snapshot might contain uncommitted transactions and full recovery may be necessary.
- VSS issues a **freeze** command to checkpoint and then suspend all activity on all databases on all database servers. Each SQL Anywhere database server waits a maximum of 60 seconds for all databases to suspend all activity. Typically, this process takes a few seconds.
- VSS issues a **thaw** command to the SQL Anywhere VSS writer to resume all transactions on all databases on all database servers.

In rare circumstances, SQL Anywhere might be unable to suspend transactions or complete a checkpoint within the maximum time allowed by VSS. If this occurs, you must use the transaction log file and the full recovery process to recover the backed up database.

See also

- "Service utility (dbsvc) for Windows" on page 840
- "Create Windows services" on page 59

Making a client-side backup

You can use the Backup utility (dbbackup) to make a backup on the client computer. The Backup utility (dbbackup) uses physical device-level parallelism to decrease the overall time required to complete a backup operation. Parallel backups are not supported on Windows Mobile. See "Understanding parallel database backups" on page 926.

The following methods are supported for making a client-side backup:

Tool	More information
Backup utility (dbbackup)	"Make a client-side backup by using the dbbackup utility" on page 899
DBBackup function	"a_backup_db structure" [*SQL Anywhere Server - Programming*]

Make a client-side backup by using the dbbackup utility

The dbbackup utility makes an image backup, which consists a copy of the database file and/or the transaction log, each as separate files.

To make a client-side backup (dbbackup utility)

- Run the Backup utility (dbbackup) on the client computer. For example:

```
dbbackup -c "HOST=myhost;DBN=demo;UID=DBA;PWD=sql" SQLAnybackup
```

See also

- "Backup utility (dbbackup)" on page 767
- "Making a server-side backup" on page 894
- "Recovering your database" on page 902
- "Types of backup" on page 888

Make a live backup

You can use a live backup to provide a redundant copy of the transaction log. This copy can be used to restart a secondary system if the primary system running the database server becomes unusable. A live backup runs continuously, terminating only if the server shuts down. If a system failure occurs, the backed up transaction log can be used for a rapid restart of the system. However, depending on the load that the server is processing, the live backup may lag behind and may not contain all committed transactions.

You should run the dbbackup utility from the secondary computer. If the primary computer becomes unusable, you can restart your database using the secondary computer. The database file and the transaction log hold the information needed to restart.

You carry out a live backup of the transaction log by using the dbbackup utility with the -l option.

To make a live backup (dbbackup utility)

1. Set up a secondary computer from which you can run the database if the online computer fails. Ensure that you have SQL Anywhere installed on the secondary computer.

2. Periodically, perform a full backup to the secondary computer.

   For example:

   ```
   dbbackup -c "UID=DBA;PWD=sql;HOST=myhost;Server=testsrv;DBN=test" c:
   \backup
   ```

3. Run a live backup of the transaction log to the secondary computer.

   ```
   dbbackup -l path\filename.log -c "connection-string"
   ```

4. Regularly run the dbbackup utility from the secondary computer.

   If the primary computer becomes unusable, the database can be restarted using the secondary computer. The database file and the transaction log hold the required information needed for a restart.

See also

- "Live backups" on page 891
- "Recovering your database" on page 902
- "Restart from a live backup" on page 906
- "Types of backup" on page 888

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Validating backups

Database file corruption may not be confirmed until the database server tries to access the affected part of the database. As part of your backup and recovery plan, you should periodically check that your database is valid by using tools such as the **Validate Database Wizard** in Sybase Central, or the Validation utility (dbvalid). You should validate your database both before and after you perform a backup. You must have VALIDATE authority to perform validation activities. See "VALIDATE authority" on page 447.

When you start a backup copy of a database to validate it, you can use the -ds database option to specify the location of dbspace files and the transaction log. This allows you to start the backed up copy of the database on the same computer as the original database while the original database is still running. See "-ds dbeng12/dbsrv12 database option" on page 254.

Depending on the options you specify, validation can include checksums, correctness of index data, and whether all table pages belong to objects in the database. Express database validation (the -fx option) does not validate data, continued row structure, or foreign key relationships.

Caution
Backup copies of the database and transaction log must not be changed in any way. If there were no transactions in progress during the backup, or if you specified BACKUP DATABASE WITH CHECKPOINT LOG RECOVER or WITH CHECKPOINT LOG NO COPY, you can check the validity of the backup database using read-only mode or by validating a copy of the backup database.

However, if transactions were in progress, or if you specified BACKUP DATABASE WITH CHECKPOINT LOG COPY, the database server must perform recovery on the database when you start it. Recovery modifies the backup copy, which is not desirable.

If you can be sure that no transactions are in progress when the backup is being made, the database server does not need to perform recovery steps. In this case, you can perform a validity check on the backup using the read-only database option. See "-r dbeng12/dbsrv12 server option" on page 217.

Tip
Using the BACKUP statement with the WAIT BEFORE START clause ensures that no transactions are in progress when you start a backup.

Validation requires exclusive access to the object being validated. For this reason, it is best to validate when there is no other activity on the database.

If a base table in the database file is corrupt, treat it as a media failure, and recover from your previous backup. If an index is corrupt, you may want to unload the database without indexes, and reload.

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Validate a database" on page 930
- "Validating the transaction log" on page 922
- "Validate a table" on page 931
- "VALIDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "Improving performance when validating databases" on page 932
- "Recovering your database" on page 902

Recovering your database

Recovery is the process of restoring your database file, transaction log, and dbspaces, and bringing the database file as up-to-date as possible with incremental transaction log files.

It is important that you validate your backup as part of your backup and recovery plan. You should only recover from a valid backup copy of the database.

The steps you need to take in the recovery process depend on whether you leave the transaction log untouched on incremental backup in your backup process. If your backup operation deletes or renames the transaction log, you may have to apply changes from several transaction logs. If your backup operation leaves the transaction log untouched, you need to use only the online transaction log in recovery.

If you have multiple transaction logs, transactions may span several transaction logs. You must apply the transaction logs in the correct order when recovering; otherwise, transactions that span multiple transaction logs are rolled back. You can specify the -ad database server option if you want the database server to determine the correct order in which to apply the transaction logs. See "Recovering a database with multiple transaction logs" on page 907.

The automatic recovery process

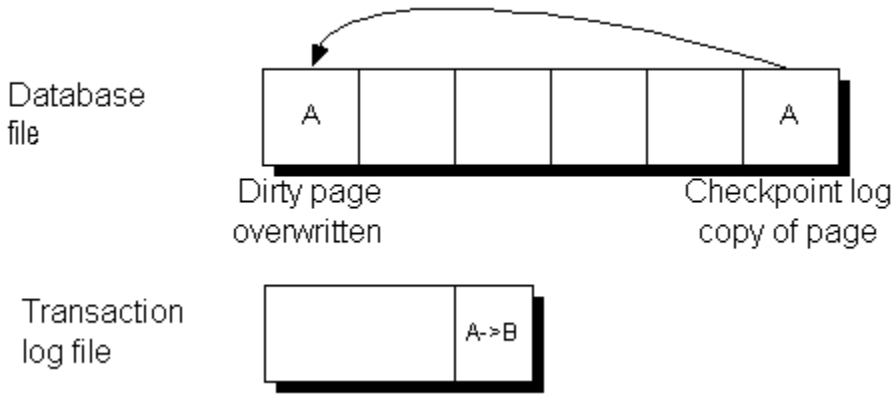
When a database is shut down during normal operation, the database server performs a checkpoint so that all the information in the database is held in the database file. This is a **clean** shutdown.

Each time you start a database, the database server checks whether the last shutdown was clean or the result of a system failure. If the database was not shut down cleanly, it automatically takes the following steps to recover from a system failure:

1. **Recover to the most recent checkpoint**

   To restore all pages to their state at the most recent checkpoint, the checkpoint log pages are copied over the changes made since the checkpoint.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0



2. **Apply changes made since the checkpoint**

   Changes made between the checkpoint and the system failure, which are held in the transaction log, are applied.

3. **Rollback uncommitted transactions**

   Any uncommitted transactions are rolled back, using the rollback logs.

Recover uncommitted operations

When recovering from media failure on the database file, the transaction log is intact. Recovery reapplies all committed transactions to the database. In some circumstances, you may want to find information about transactions that were incomplete at the time of the failure.

The **Translate Log File Wizard** helps you translate a log file into a *.sql* file from Sybase Central. You can also use the dbtran utility to translate a log file into a *.sql* file.

To recover uncommitted operations from a transaction log (Sybase Central)

1. Choose **Tools** » **SQL Anywhere 12** » **Translate Log File**.
2. Follow the instructions in the wizard.
3. Edit the translated log (SQL command file) in a text editor and identify the instructions you need.

To recover uncommitted operations from a transaction log (command line)

1. Run dbtran to convert the transaction log into a SQL command file, using the -a option to include uncommitted transactions. For example, the following command uses dbtran to convert a transaction log:

   ```
   dbtran -a sample.log changes.sql
   ```

2. Edit the translated log (SQL command file) in a text editor and identify the instructions you need.

   For more information about the Log Translation utility, see "Log Translation utility (dbtran)" on page 820.

Note
The transaction log may or may not contain changes right up to the point where a failure occurred. It does contain any changes made before the end of the most recently committed transaction that made changes to the database.

See also

- "Restore from an image backup" on page 904
- "Restore from an archive backup" on page 904
- "Restart from a live backup" on page 906
- "Recovering a database with multiple transaction logs" on page 907
- "Recovering from media failure" on page 910

Restore from an image backup

The following procedure assumes that you do not have any incremental backups of the transaction log that you need to apply as part of the recovery process. For information about recovering a database when you have backed up multiple copies of the transaction log, see "Recovering a database with multiple transaction logs" on page 907.

To restore a database from an image backup

1. Copy the database files back to their original location.
2. Restart the database server.

See also

- "Image backups" on page 893
- "The automatic recovery process" on page 902
- "Recover uncommitted operations" on page 903
- "Types of backup" on page 888

Restore from an archive backup

The following procedure assumes that you do not have any incremental backups of the transaction log that you need to apply as part of the recovery process. For information about recovering a database when you have backed up multiple copies of the transaction log, see "Recovering a database with multiple transaction logs" on page 907.

If you created the backup of a strongly-encrypted database with free page elimination turned on, you must specify the encryption key for the database when restoring it.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To restore a database from an archive backup (Sybase Central)

1. Start a personal database server.

   For example, the following command starts a database server named restore:

   ```
   dbeng12 -n restore
   ```

2. Start Sybase Central and connect to the utility database. Complete the following fields:
   - **User ID** Type **DBA**.
   - **Password** Type **sql**.
   - **Database Name** Type **utility_db**.
3. Click **OK**.
4. Choose **Tools** » **SQL Anywhere 12** » **Restore Database**.
5. Follow the instructions in the wizard.

To restore a database from an archive backup (Interactive SQL)

1. Start a personal database server.

   For example, the following command starts a database server named restore:

   ```
   dbeng12 -n restore
   ```

2. Start Interactive SQL and connect to the utility database. Complete the following fields:
   - **User ID** Type **DBA**.
   - **Password** Type **sql**.
   - **Database Name** Type **utility_db**.
3. Click **OK**.
4. Execute the RESTORE DATABASE statement, specifying the archive root.

   At this time, you can choose to restore an archived database to its original location (the default), or to a different computer with different device names using the RENAME clause. See "RESTORE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

Example

The following statement restores a database from a tape archive to the database file *c:\newdb\newdb.db*.

```
RESTORE DATABASE 'c:\\newdb\\newdb.db'
FROM '\\\\.\\tape0';
```

The following statement restores a database from an archive backup in file *c:\backup\archive.1* to the database file *c:\newdb\newdb.db*. The transaction log name and location are specified in the database.

```
RESTORE DATABASE 'c:\\newdb\\newdb.db'
FROM 'c:\\backup\\archive';
```

The following statement restores a database from an archive backup in file *c:\backup\archive.1* to the database file *c:\newdb\newdb.db*. The transaction log name and location are specified in the database. The encryption key is specified for the database.

```
RESTORE DATABASE 'c:\\newdb\\newdb.db'
FROM 'c:\\backup\\archive'
KEY '3Km57y1z';
```

See also

- "Using the utility database" on page 28
- "Archive backups" on page 892
- "The automatic recovery process" on page 902
- "Recover uncommitted operations" on page 903
- "Types of backup" on page 888

Restart from a live backup

A live backup is made to a separate computer from the primary computer that is running your production database. To restart a database from a live backup, you must have SQL Anywhere installed on the secondary computer. For more information about live backups, see "Live backups" on page 891.

To restart a database using a live backup

1. Copy the full backup transaction log file and the live backup transaction log to a directory where they can be applied to the backup copy of the database file.
2. Rename or delete the current transaction log file whose name matches the expected transaction log file name, if one exists.
3. Start the database server with the -ad option to apply the transaction logs in the directory created in step 1 and bring the database up to date:

   ```
   dbeng12 samples-dir\demo.db -ad directory-name
   ```

   The database server shuts down automatically once the transaction log is applied.
4. Start the database server in the normal way, allowing user access. Any new activity is written to a new transaction log.
5. Run a live backup of the transaction log to the secondary computer.

   ```
   dbbackup -l path\filename.log -c "connection-string"
   ```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Live backups" on page 891
- "The automatic recovery process" on page 902
- "Recover uncommitted operations" on page 903
- "Types of backup" on page 888

Recovering a database with multiple transaction logs

If you need to recover your database and you have multiple transaction logs, you must apply the transaction log files to the backup copy of your database in the correct order.

You can use any of the following methods to apply transaction logs in the correct order:

- Use the -a server option to apply each log individually to the backup copy of the database. You can use the Transaction Log utility (dblog) to determine the order in which transaction log files were generated. The utility generates and displays the earliest log offset in the transaction log, which can be an effective method for determining the order in which to apply multiple log files. See "-a dbeng12/dbsrv12 database option" on page 251.
- Use the -ad server option to specify the location of the transaction log files. The database server determines the correct order for applying the transaction logs to the backup copy of the database based on the log offsets. See "-ad dbeng12/dbsrv12 database option" on page 252.
- Use the -ar server option to have the database server apply log files associated with the database that are located in the same directory as the transaction log. The transaction log location is obtained from the database. The database server determines the correct order for applying the transaction logs to the backup copy of the database based on the log offsets. See "-ar dbeng12/dbsrv12 database option" on page 253.
- Use the Log Translation utility (dbtran) to translate one or more transaction logs into a *.sql* file that can be applied to the backup copy of the database. See "Transaction Log utility (dblog)" on page 862.

Recover a database with multiple transaction logs using the -ad server option

The -ad server option is used to recover a database by applying all the transaction logs from a specified directory to the backup copy of a database. When this option is specified, the database server applies the transaction logs and then shuts down the database.

To recover from multiple transaction logs using the -ad server option

- Start the database server using -ad to apply the transaction logs to the backup copy of your database. See "-ad dbeng12/dbsrv12 database option" on page 252.

Example

The following example applies the offline (backup) and current transaction logs to the backup copy of the sample database using the -ad database server option. The database server uses the log offsets in the transaction logs to determine the correct order in which to apply the log files.

1. Copy the backup transaction log and current transaction log into a directory, for example, *c:\backuplogs*.

2. Start the database server and apply the transaction logs to a backup copy of a database called *backupdemo.db*:

   ```
   dbeng12 backupdemo.db -ad c:\backuplogs
   ```

   The database server applies the transaction logs to the backup copy of the database and then shuts down.

See also

- "Recover a database with multiple transaction logs using the -a server option" on page 908
- "Recover a database with multiple transaction logs using the dbtran utility" on page 909
- "The automatic recovery process" on page 902
- "Recover uncommitted operations" on page 903

Recover a database with multiple transaction logs using the -a server option

The -a server option is used recover a database by applying a single transaction log file to the backup copy of a database. When this option is specified, the database server applies the log and then shuts down. If you have multiple transaction logs, you must apply them one at a time in the correct order, from oldest to most recent.

To recover from multiple transaction logs using the -a server option

1. Start the database server using -a to apply the backup transaction log to the offline (backup) copy of your database.

   See "-a dbeng12/dbsrv12 database option" on page 251.

2. Start the database server and apply the current transaction log to the backup copy of your database.

Example

The following example applies the offline (backup) and current transaction logs to the backup copy of the sample database using the -a database server option.

1. Start the database server and apply a backup transaction log called *backupdemo.log* to the backup copy of a database called *backupdemo.db*:

   ```
   dbeng12 backupdemo.db -a backupdemo.log
   ```

   The database server applies the backup transaction log to the backup copy of the database and then shuts down.

2. Start the database server and apply the current transaction log called *demo.log* to the backup copy of the database:

   ```
   dbeng12 backupdemo.db -a demo.log
   ```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The database server applies the current transaction log to the backup copy of the database and then shuts down.

See also

- "Recover a database with multiple transaction logs using the -ad server option" on page 907
- "Recover a database with multiple transaction logs using the dbtran utility" on page 909
- "The automatic recovery process" on page 902
- "Recover uncommitted operations" on page 903

Recover a database with multiple transaction logs using the dbtran utility

To maintain the integrity of your data when you use dbtran to translate multiple transaction logs, you must specify both the -m and -n options. The -m option instructs the Log Translation utility (dblog) to generate a file (named by -n) containing all the transactions from the logs in the specified directory.

You need to use -m because any transactions that span transaction log files could be rolled back if you translate each log individually using dbtran. When dbtran translates a log, it adds a ROLLBACK statement to the end of the log to undo any uncommitted transactions. When a transaction spans two logs, the COMMIT for the transaction occurs in the second log file. Operations at the end of the first log file would be rolled back by dbtran because the file does not contain a COMMIT for the transaction. Translating all the transaction log files in a directory using -m ensures that all of your transactions are translated. See "Transaction Log utility (dblog)" on page 862.

To recover from multiple transaction logs (dbtran utility)

1. Run the Log Translation utility (dbtran) against the directory containing the transaction log files and output the resulting SQL statements into a *.sql* file.
2. Start the backup copy of your database.
3. Apply the *.sql* file generated by dbtran in step 1 to the backup copy of your database from Interactive SQL.

Example

The following example uses the dbtran utility to apply the backup and current transaction logs to the backup copy of the database.

1. Run the Log Translation utility against the *c:|backup* directory and output the SQL statements into a file called *recoverylog.sql*:

   ```
   dbtran -m "c:\backup" -n recoverylog.sql
   ```

2. Start the backup copy of the database called *backupdemo.db*:

   ```
   dbeng12 backupdemo.db
   ```

3. Apply the *recoverylog.sql* file to the database from Interactive SQL:

   ```
   dbisql -c "UID=DBA;PWD=sql;Server=backupdemo" READ recoverylog.sql
   ```

See also

- "Recover a database with multiple transaction logs using the -ad server option" on page 907
- "Recover a database with multiple transaction logs using the -a server option" on page 908
- "The automatic recovery process" on page 902
- "Recover uncommitted operations" on page 903

Recovering from media failure

If your database is unusable, you have experienced a database **failure**. SQL Anywhere provides protection against the following types of failure:

Type of failure	Description	Examples
Media	The database file and/or the transaction log become unusable. This type of failure can occur because the file system or the device storing the database file becomes unusable, or because of file corruption. Backups protect your data against media failure.	• The disk drive storing the database file or the transaction log file becomes unusable. • The database file or the transaction log file becomes corrupt. This can happen because of hardware or software problems.
System	A system failure occurs when the computer or operating system fails while there are partially completed transactions. This type of failure can occur when the computer is inappropriately turned off or restarted, when another application causes the operating system to fail, or because of a power failure. After a system failure occurs, the database server recovers automatically when you next start the database. The results of each transaction committed before the system error are intact. All changes by transactions that were not committed before the system failure are canceled.	• The computer or operating system becomes temporarily unavailable while there are partially completed transactions, perhaps because of a power failure or operating system failure, or because the computer is inappropriately restarted.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Recover from media failure on the data

This procedure describes the steps for recovering from media failure if the only file that you lost is the database.

To recover from media failure on the database file

1. Make an extra backup copy of the current transaction log. Since the database file is unavailable, the transaction log contains the only record of the changes that have been made since the last backup.
2. Create a **recovery directory** to hold the files you use during recovery.
3. Copy the database file from the last full backup to the recovery directory.
4. Apply the transactions held in the backed up transaction logs to the recovery database. Use one of the following methods.

   To apply each transaction log manually, for each log file, chronologically, do the following:

   a. Copy the log file into the recovery directory.

   b. Start the database server with the apply transaction log (-a) option, to apply the transaction log:

      ```
      dbeng12 database-name.db -a log-name.log
      ```

      The database server shuts down automatically once the transactions are applied.

   c. Once you have applied all the backed up transaction logs, copy the online transaction log into the recovery directory.

      Apply the transactions from the online transaction log to the recovery database.

      ```
      dbeng12 database-name.db -a log-name.log
      ```

   If you want the database server to determine the correct order of the transaction logs and apply them automatically, do the following:

   a. Copy the offline and online transaction log files into the recovery directory.

   b. Start the database server with the -ad option, to specify the location of the transaction logs. The database server determines the correct order in which to apply the transaction logs based on the log offsets:

      ```
      dbeng12 database-name.db -ad log-directory
      ```

      The database server shuts down automatically once the transactions are applied.

5. Perform validity checks on the recovery database.

   See “Validate a database” on page 930.

6. Make a backup.
7. Move the database file to the production directory.
8. Notify users that they can access the production database.

See also

- "The automatic recovery process" on page 902
- "Recover uncommitted operations" on page 903

Recover from media failure on a transaction log mirror

The following procedure explains how to recover from a media failure when you are using a transaction log mirror. If your database is a consolidated database in a SQL Remote installation, you should use a transaction log mirror, or a hardware equivalent.

To recover from media failure on a transaction log mirror

1. Make an extra copy of the backup of your database file taken at the time the transaction log was started.
2. Identify which of the two files is corrupt. Run the Log Translation utility (dbtran) on the transaction log and on its mirror. The file that generates an error message is corrupt. The Log Translation utility is accessible from Sybase Central or as the dbtran utility.

   The following command line translates a transaction log named *demo.log*, placing the translated output into *demo.sql*:

   ```
   dbtran demo.log
   ```

   The Log Translation utility properly translates the intact file, and reports an error while translating the corrupt file.
3. Copy the correct file over the corrupt file.
4. Restart the database server.

See also

- "Transaction log mirrors" on page 22
- "The automatic recovery process" on page 902
- "Recover uncommitted operations" on page 903

Recover from media failure on an unmirrored transaction log

If your database is a consolidated database in a MobiLink or SQL Remote installation, you should use a transaction log mirror, or hardware equivalent. See "Transaction log mirrors" on page 22.

To recover from media failure on an unmirrored transaction log (partial recovery)

1. Make an extra backup copy of the database file. Without a transaction log, the database file contains the only record of the changes made since the last backup and the most recent checkpoint.
2. Delete or rename the transaction log file.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

3. Restart the database with the -f option.

   ```
   dbeng12 samples-dir\demo.db -f
   ```

   Caution
   This command should only be used when the database is not participating in a MobiLink or SQL Remote system. If your database is a consolidated database in a SQL Remote replication system, you may have to re-extract the remote databases.

   Without the -f option, the database server reports the lack of a transaction log as an error. With the -f option, the database server restores the database to the most recent checkpoint and then rolls back any transactions that were not committed at the time of the checkpoint. A new transaction log is then created.

See also

- "The transaction log" on page 21
- "The automatic recovery process" on page 902
- "Recover uncommitted operations" on page 903
- "-f dbeng12/dbsrv12 server recovery option" on page 180

Designing a backup and recovery plan

It is recommended that you develop and implement a backup schedule to protect your data. You should also ensure that you have created and tested your backup and recovery commands as part of your backup and recovery plan.

Some of the factors that you need to consider when developing your backup and recovery plan include:

- where are the database files located?
- what files need to be backed up?
- where are the backup files stored?
- how does the backup affect performance of your database or application?
- will the database server be running while you run the backup?

Some of the most common situations where you require a backup include:

- failed media
- failed hardware
- file corruption

Typically, a backup uses a combination of full and incremental backups. The frequency of each backup type depends on the type of data that you are protecting. You should also validate your backups to ensure that they can be used for recovery. See "Validating backups" on page 901.

You can use the scheduling features in SQL Anywhere to automate the task of backing up your database. Once you specify a schedule, the backups are performed automatically by the database server. See

"Automating tasks using schedules and events" on page 932 and "Creating a maintenance plan" on page 915.

The length of time your organization can function without access to the data in your database determines the maximum recovery time.

You should verify that you have the protection you need against media failure on the database file and on the transaction log file. If you are running in a replication environment, you should consider using a transaction log mirror. See "Protecting against media failure" on page 917.

External factors such as available hardware, the size of database files, recovery medium, disk space, and unexpected errors can affect your recovery time. When planning a backup strategy, you should allow additional recovery time for tasks such as entering recovery commands or retrieving and loading tapes.

See also

- "Understanding backups" on page 923
- "Backing up databases involved in synchronization and replication" on page 918
- "Backup and recovery restrictions" on page 893
- "Types of backup" on page 888
- "Choosing a backup format" on page 892
- "Full backups" on page 889
- "Incremental backups" on page 890

Implement a backup and recovery plan

To implement a backup and recovery plan

1. Create and verify your backup and recovery commands, including commands for database validation. See "Validating backups" on page 901.
2. Measure the time it takes to execute backup and recovery commands.
3. Document the backup commands and create written procedures outlining where your backups are kept. The procedures should identify any naming conventions that are used, and the type of backups that are performed.
4. Set up your backup procedures on the production server.
5. Monitor backup procedures to avoid unexpected errors. Make sure any changes in the process are reflected in your documentation.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- “Full backups” on page 889
- “Incremental backups” on page 890
- “Understanding backups” on page 923
- “Backing up databases involved in synchronization and replication” on page 918
- “Backup and recovery restrictions” on page 893
- “Types of backup” on page 888
- “Choosing a backup format” on page 892

Scheduling considerations

Typically, a backup uses a combination of full and incremental backups. The frequency with which you make backups depends on such factors as the importance of your data and how often it changes.

A common starting point for backups is to perform a weekly full backup, with daily incremental backups of the transaction log. Both full and incremental backups can be performed online (while the database is running) or offline, on the server side or the client side.

The kinds of failure against which a backup schedule provides protection is dependent not only the frequency of your backup schedule, but also on how you operate your database server.

You should always keep more than one full backup. You should also keep some of your full backups off site to protect against fire, flood, earthquake, theft, or vandalism.

You can use the event scheduling features of SQL Anywhere to perform online backups automatically at scheduled times. See “Creating a maintenance plan” on page 915.

Caution
If you make a backup on top of a previous backup, a media failure in the middle of the backup leaves you with no backup at all.

Creating a maintenance plan

To simplify administration, you can set up a maintenance plan for your database that is executed automatically by the database server. A maintenance plan consists of a schedule for performing one or more of the following tasks:

- validating the database
- backing up the database
- managing maintenance plan reports

In Sybase Central you create a maintenance plan by using the **Create Maintenance Plan Wizard**. Only one instance of a maintenance plan can run at a time. Each time the maintenance plan runs, a maintenance plan report is saved in the database. You can view this report from Sybase Central and optionally you can have the maintenance plan report emailed to you after the maintenance plan executes on the database.

Customizing maintenance plans

Maintenance plans can contain user-defined operations. In the **Create Maintenance Plan Wizard** you can add user-defined operations as SQL statements that run either before validation or after backup.

Understanding the maintenance plan status

When a maintenance plan is running, its status appears in the **Maintenance Plans** folder in Sybase Central. A running maintenance plan has one of the following statuses:

- **Initializing** Disables new connections and disables current users.
- **Running prevalidation** Runs user-defined operations (if any were specified when the maintenance plan was created).
- **Validating pages** Indicates the status of database page validation.
- **Validating objects** Indicates the status of table, materialized view, and index validation.
- **Backing up** Indicates the progress of the backup.
- **Running post-backup** Runs user-defined operations (if any were specified when the maintenance plan was created).
- **Terminating** Re-enables new connections, and saves, discards, and emails report.

Create a maintenance plan report

To create a maintenance plan (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, right-click **Maintenance Plans** and choose **New** » **Maintenance Plan**.
3. Follow the instructions in the wizard.

   For information about the available settings, see:

   - "Defining schedules" on page 935
   - "VALIDATE statement" [*SQL Anywhere Server - SQL Reference*]
   - "Types of backup" on page 888
   - "xp_startsmtp system procedure" [*SQL Anywhere Server - SQL Reference*]

See also

- "Archive backups" on page 892
- "Image backups" on page 893
- "Incremental backups" on page 890
- "Creating a maintenance plan" on page 915

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

View the maintenance plan report

After the maintenance plan has executed, you can view a report in Sybase Central.

To view the maintenance plan report (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, double-click **Maintenance Plans**.
3. Double-click your maintenance plan.
4. In the right pane, double-click the report.

   The **Maintenance Plan Properties** window appears. The **Details** pane contains the log for the maintenance plan.

Protecting against media failure

Backups protect your data against media failure.

When you create a database, the default location for the transaction log is the same device and in the same directory as the database file. This arrangement does not protect against media failure, and you should consider placing the transaction log in another location for production use.

Media failure on the database file If your database file is unusable and your transaction log is usable, you can recover all committed changes to the database as long as you have a proper backup procedure in place. All information since the last backed up copy of the database file is held in backed up transaction logs, or in the online transaction log.

Media failure on the transaction log file Unless you use a transaction log mirror, you cannot recover information entered between the last database checkpoint and a media failure on the transaction log. For this reason, it is recommended that you use a transaction log mirror in setups such as SQL Remote consolidated databases, where loss of the transaction log can lead to loss of key information, or the breakdown of a replication system.

How quickly you can recover from media failure depends on whether the media failure is on the database file or the transaction log file.

For comprehensive protection against media failure, you should keep the transaction log on a different device from the database file. Some computers with two or more hard drives have only one physical disk drive with several logical drives or partitions: if you want reliable protection against media failure, make sure that you have a computer with at least two physical storage devices.

Placing the transaction log on a separate device can also improve performance by eliminating the need for disk head movement between the transaction log and the main database file.

Caution
You should not place the transaction log on a network directory. Reading and writing pages over a network results in poor performance and possible file corruption.

See also

- "Creating a SQL Anywhere database" on page 5
- "Change the location of a transaction log" on page 23

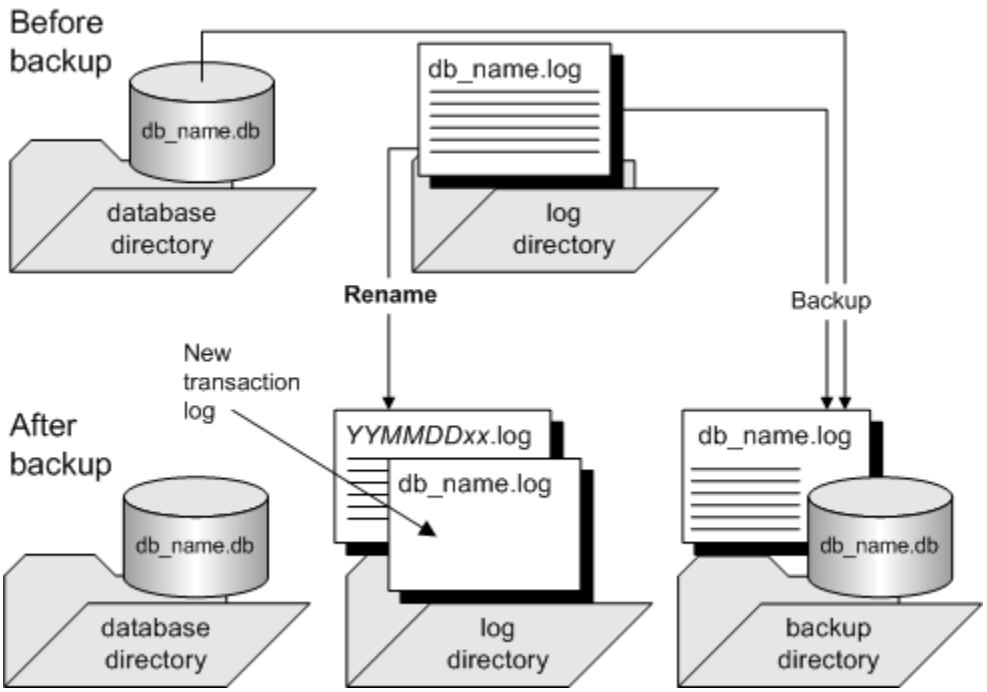
Backing up databases involved in synchronization and replication

If your database is part of a SQL Remote installation, the Message Agent must have access to old transactions. If it is a consolidated database, it holds the master copy of the entire SQL Remote installation, and thorough backup procedures are essential to ensure that no data is lost.

If your database is participating in a MobiLink setup using dbmlsync, the same considerations apply. However, if your database is a MobiLink consolidated database, old transaction logs are not required.

For synchronization and replication environments, you can choose backup options to rename and restart the transaction log. This kind of backup prevents open-ended growth of the transaction log, while maintaining information about the old transactions.

This kind of backup is illustrated in the figure below.



For more information, see "Make a backup and rename the original transaction log" on page 919.

Backup procedures are not as crucial on remote databases as they are on the consolidated database. You may choose to rely on replication to the consolidated database as a data backup method. In the event of a

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

media failure, the remote database would have to be re-extracted from the consolidated database, and any operations that have not been replicated would be lost. You could use the Log Translation utility to attempt to recover lost operations. See "Log Translation utility (dbtran)" on page 820.

Even if you do choose to rely on replication to protect remote database data, backups may still need to be done periodically at remote databases to prevent the transaction log from growing too large. You should use the same option (rename and restart the log) as at the consolidated database, running the Message Agent so that it has access to the renamed log files. If you set the delete_old_logs option to On at the remote database, the old log files are deleted automatically by the Message Agent when they are no longer needed.

Automatic transaction log renaming in SQL Remote

Use the -x Message Agent option to eliminate the need to rename the transaction log on the remote computer when the database server is shut down. The -x option renames the transaction log after it has been scanned for outgoing messages. See "SQL Remote Message Agent utility (dbremote)" [*SQL Remote*].

Managing the transaction log

When you back up your database, you must decide whether you want to continue to use the existing transaction log or create a new one. If your database is involved in synchronization or replication, you must maintain copies of old transaction logs until you are certain that they are no longer needed.

Make a backup and rename the original transaction log

To make a backup, renaming the transaction log (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with BACKUP authority.
2. Right-click the database and choose **Create Backup Images**.
3. Click **Next**.
4. In the **Which Database Do You Want To Back Up** list, select the database and click **Next**.
5. In the **Save The Backup Images In The Following Directory** field, type the name of a directory to save the backup copies.
6. Select an option in the **Which Files Do You Want To Back Up** list and click **Next**.
7. In the **What Do You Want To Do With The Transaction Log** list, click **Rename The Transaction Log**.
8. Click **Next**.
9. Click **Finish**.

10. Click **Close**.

To make a backup, renaming the transaction log (SQL)

- Use the BACKUP statement with the following clauses:

```
BACKUP DATABASE
DIRECTORY backup-directory
[ TRANSACTION LOG ONLY ]
TRANSACTION LOG RENAME;
```

  Include the TRANSACTION LOG ONLY clause only if you are making an incremental backup.

  The backup copies of the transaction log and database file are placed in *backup-directory*. If you enter a path, it is relative to the working directory of the database server, not your client application.

To make a backup, renaming the transaction log (command line)

- Run the following command (it must be typed on one line):

```
dbbackup -c "connection-string" -r [ -t ] backup-directory
```

  Include the -t option if you are making an incremental backup.

  The backup copies of the transaction log and database file are placed in *backup-directory*. If you enter a path, it is relative to the directory from which you run the command.

See also

- "Backup utility (dbbackup)" on page 767
- "BACKUP statement" [*SQL Anywhere Server - SQL Reference*]
- "The transaction log" on page 21
- "Recovering your database" on page 902

Rename the backup copy of the transaction log during backup

By default, the backup copy of the transaction log file has the same name as the online file. For each backup operation, you must assign a different name or location for the backup copy, or you must move the backup copy before the next backup is done.

To make a repeatable incremental backup command, rename the backup copy of the transaction log.

To rename the backup copy of the transaction log (SQL)

- Use the MATCH keyword in the BACKUP statement. For example, the following statement makes an incremental backup of the transaction log to the directory *c:\backup*. The backup copy of the transaction log is called *YYMMDDxx.log*, where *YYMMDD* is the date and *xx* is a counter, starting from AA.

```
BACKUP DATABASE
DIRECTORY 'c:\\backup'
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME MATCH;
```

To rename the backup copy of the transaction log (command line)

- Supply the -n option to dbbackup. For example, the following command makes an incremental backup of the sample database, renaming the backup copy of the transaction log.

  ```
  dbbackup -c "UID=DBA;PWD=sql;DBN=demo" -r -t -n c:\backup
  ```

Notes

The backup copy of the transaction log is named *YYMMDDxx.log*, where *YY* is the year, *MM* is the month, *DD* is the day of the month, and *xx* runs from AA to ZZ, incrementing if there is more than one backup per day. The *YYMMDDxx.log* file names are used to distinguish between files, not for ordering.

This set of backup options is typically used for databases involved in replication. In addition to making backup copies of the database file and transaction log, the transaction log at backup time is renamed to an offline log, and a new transaction log is started with the same name as the log in use at backup time.

See also

- "Backing up databases involved in synchronization and replication" on page 918
- "Backup utility (dbbackup)" on page 767
- "BACKUP statement" [*SQL Anywhere Server - SQL Reference*]
- "The transaction log" on page 21
- "Recovering your database" on page 902

Make a backup and delete the original transaction log

If your database is not involved in replication and you have limited disk space on your computer, you can delete the contents of the online transaction log (**truncate** the log) when you make a backup. To recover your database when using this type of backup, you must use every backup copy made since the last full backup during recovery from media failure on the database file.

To make a backup, deleting the transaction log (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Right-click the database and choose **Create Backup Images**.
3. Click **Next**.
4. In the **Which Database Do You Want To Back Up** list, select the database and click **Next**.
5. In the **Save The Backup Images In The Following Directory** field, type the name of a directory to save the backup copies.
6. Select an option in the **Which Files Do You Want To Back Up** list and click **Next**.

7. In the **What Do You Want To Do With The Transaction Log** list, click **Truncate The Transaction Log**.
8. Click **Next**.
9. Click **Finish**.
10. Click **Close**.

To make a backup, deleting the transaction log (SQL)

- Use the BACKUP statement with the following clauses:

```
BACKUP DATABASE
DIRECTORY backup-directory
[ TRANSACTION LOG ONLY ]
TRANSACTION LOG TRUNCATE;
```

  Include the TRANSACTION LOG ONLY clause only if you are making an incremental backup.

  The backup copies of the transaction log and database file are placed in *backup-directory*. If you enter a path, it is relative to the working directory of the database server, not your client application.

To make a backup, deleting the transaction log (command line)

- Run the following command:

```
dbbackup -c "connection-string" -x [ -t ] backup-directory
```

  Include the -t option only if you are making an incremental backup.

  The backup copies of the transaction log and database file are placed in *backup-directory*. If you enter a path, it is relative to the directory from which you run the command.

See also

- “Backup utility (dbbackup)” on page 767
- “BACKUP statement” [*SQL Anywhere Server - SQL Reference*]
- “The transaction log” on page 21
- “Recovering your database” on page 902

Validating the transaction log

When a database using a transaction log mirror starts up, the database server performs a series of checks and automatic recovery operations to confirm that the transaction log and its mirror are not corrupt, and to correct some problems if corruption is detected.

On startup, the server checks that the transaction log and its mirror are identical by performing a full comparison of the two files; if they are identical, the database starts as usual. The comparison of log and mirror adds to database startup time.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

If the database stopped because of a system failure, some operations might be written into the transaction log but not into the mirror. If the server finds that the transaction log and the mirror are identical up to the end of the shorter of the two files, the remainder of the longer file is copied into the shorter file. This produces an identical log and mirror. After this automatic recovery step, the server starts as usual.

If the check finds that the transaction log and the transaction log mirror are different in the body, one of the two files is corrupt. In this case, the database does not start, and an error message is generated saying that the transaction log or its mirror is invalid.

You can also use the Log Translation utility (dbtran) to validate transaction logs whether you have an online or offline transaction log. If the Log Translation utility can successfully read the log file, it is valid. See "Log Translation utility (dbtran)" on page 820.

The internal backup process

When you start a backup, the database may be in use by many people. If you need to restore your database from a backup, you need to know what information has been backed up, and what has not.

When making a backup, the database server:

1. Issues a checkpoint. Further checkpoints are disallowed until the backup is complete.
2. Makes a backup of the database files, if performing a full backup.
3. Makes a backup of the transaction log.

   The backup includes all operations recorded in the transaction log before the final page of the log is read. This may include instructions issued after the backup started.

   The backup copy of the transaction log is generally smaller than the online transaction log. The database server allocates space to the online transaction logs in multiples of 64 KB, so the transaction log file size generally includes empty pages. However, only the non-empty pages are backed up.
4. Marks the backup image of the database to indicate that recovery is needed. This step causes any operations that happened since the start of the backup to be applied when the backup copy of the database is started. It also causes operations that were incomplete at the checkpoint to be undone if they were not committed.

Understanding backups

When a database shuts down cleanly, the database file holds a complete and current copy of all the data in the database. When a database is running, however, the database file is generally not current or complete.

The only time a database file is guaranteed to hold a complete and current copy of all data is immediately after a checkpoint completes. Following a checkpoint, all the contents of the database cache are on disk.

The database server checkpoints a database under the following conditions:

- As part of the database shutdown operations
- When the amount of time since the last checkpoint exceeds the setting of the -gc server option
- When the estimated time to do a recovery operation exceeds the setting of the -gr server option
- When the database server is idle long enough to write all dirty pages
- When certain DDL statements (such as ALTER TABLE, DROP TABLE, DROP INDEX, LOAD TABLE, or BACKUP) are executed
- When a connection issues a CHECKPOINT statement
- When the database server is running without a transaction log and a transaction is committed

To ensure that you have a complete copy of all committed transactions between checkpoints, you need the database file and the transaction log.

See also

- "Understanding the checkpoint log" on page 25
- "How the database server decides when to checkpoint" on page 924
- "-gc dbeng12/dbsrv12 server option" on page 184
- "-gr dbeng12/dbsrv12 server option" on page 194

How the database server decides when to checkpoint

The priority of writing dirty pages to the disk increases as the time and the amount of work since the last checkpoint increases. The priority is determined by the following factors:

- **Checkpoint Urgency** The time that has elapsed since the last checkpoint, as a percentage of the checkpoint time setting of the database. You can set the maximum time, in minutes, between checkpoints by using the -gc server option or the checkpoint_time database option. If -gc is specified, the checkpoint_time option setting in the database is ignored.
- **Recovery Urgency** A heuristic to estimate the amount of time required to recover the database if it fails right now. You can set the maximum time, in minutes, for recovery in the event of system failure by using the -gr server option or recovery_time database option. If -gr is specified, the recovery_time option setting in the database is ignored.

The checkpoint and recovery urgency values are important only if the database server does not have enough idle time to write dirty pages. The lower boundary on the interval between checkpoints is based on a combination of the recovery_time and checkpoint_time options. The recovery_time option setting is not respected when it would force a checkpoint too soon.

Frequent checkpoints make recovery quicker, but also create work for the server writing out dirty pages.

If, because of other activity in the database, the number of dirty pages falls to zero, and if the checkpoint urgency is 33% or more, then a checkpoint takes place automatically since it is a convenient time.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Both the checkpoint urgency and recovery urgency values increase until the checkpoint occurs, at which point they drop to zero.

See also

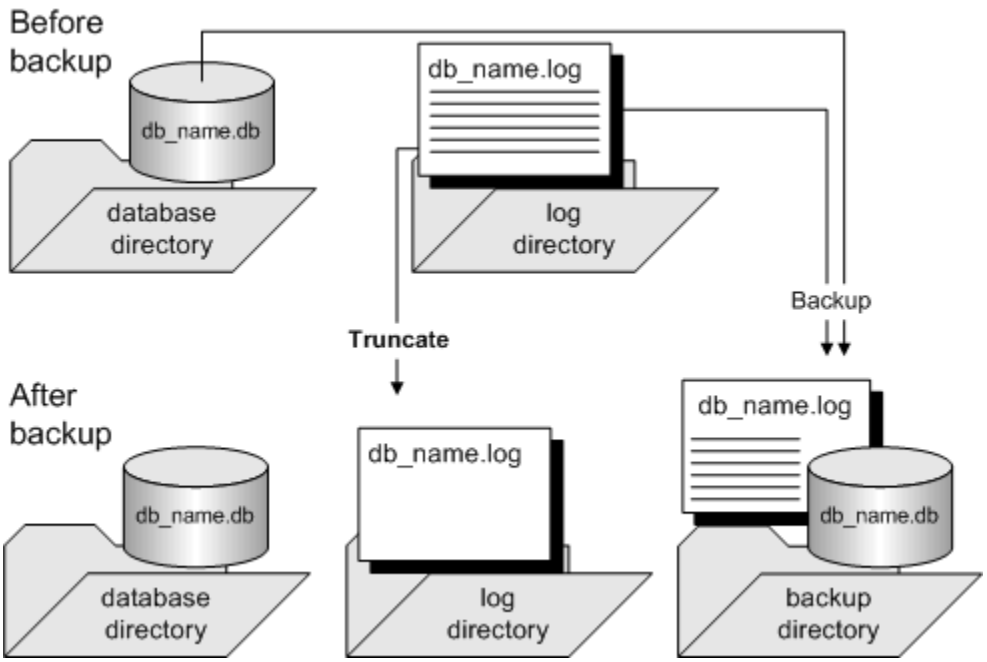
- "Understanding the checkpoint log" on page 25
- "-gc dbeng12/dbsrv12 server option" on page 184
- "checkpoint_time option" on page 519
- "-gr dbeng12/dbsrv12 server option" on page 194
- "recovery_time option" on page 580

Managing the transaction log

When you make a backup, by default the backup makes a copy of the current state of the transaction log, and leaves the transaction log in place. If your database is involved in synchronization or replication, then you may need to access old copies of the transaction log after recovering your database.

In many circumstances, disk space limitations make it impractical to let the transaction log grow indefinitely. To free disk space, you can choose to delete the contents of the transaction log when the backup is complete, freeing the disk space. Do not choose this option if the database is involved in replication because replication requires access to the transaction log.

A full backup, which truncates the log file, is illustrated in the figure below. In an incremental backup, only the transaction log is backed up.



Deleting the transaction log after each incremental backup makes recovery from a media failure on the database file a more complex task. Each transaction log needs to be applied in sequence to bring the database up to date, and there may then be several different transaction logs since the last full backup.

You can use this kind of backup on a database that is operating as a MobiLink consolidated database because MobiLink does not rely on the transaction log. If you are running SQL Remote or the MobiLink *dbmlsync.exe* application, you must use a scheme suitable for preserving old transaction logs.

See also

- "Make a backup and rename the original transaction log" on page 919
- "Rename the backup copy of the transaction log during backup" on page 920
- "Make a backup and delete the original transaction log" on page 921

Offline transaction logs

In addition to backing up the transaction log, a backup operation can rename the online transaction log to a file name of the form *YYMMDDxx.log*. This file is no longer used by the database server, but is available for the Message Agent. It is called an **offline** transaction log. A new online transaction log is started with the same name as the old online transaction log.

The *YYMMDDxx.log* file names are used to distinguish between the files, not for ordering. For example, the renamed log file from the first backup on December 10, 2000, is named *001210AA.log*. The first two digits indicate the year, the second two digits indicate the month, the third two digits indicate the day of the month, and the final two characters distinguish among different backups made on the same day.

The Message Agent can use the offline copies to provide the old transactions as needed. If you set the delete_old_logs database option to On, then the Message Agent deletes the offline files when they are no longer needed, saving disk space.

The rollback log

As changes are made to the contents of a database, a **rollback log** is created so that changes can be canceled if a transaction is rolled back or if a transaction is uncommitted when a system failure occurs. There is a separate rollback log for each connection. When a transaction is committed or rolled back, the contents of the rollback log for that connection are deleted. The rollback logs are stored in the database, and rollback log pages are copied into the checkpoint log along with other pages that are changed.

The rollback log is also called the **undo log**.

For more information about transaction processing, see "Using transactions and isolation levels" [*SQL Anywhere Server - SQL Usage*].

Understanding parallel database backups

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

When you perform a server-side image backup using the Backup utility (dbbackup) by specifying the -s option, or by using the BACKUP DATABASE statement, a parallel database backup is performed. Parallel backups use physical device-level parallelism to decrease the overall time required to complete a backup operation. Parallel backups are not supported on Windows Mobile.

The database server creates a reader thread for each drive on which database files are stored. A writer thread is created for the destination drive where the backup directory is located. Using separate readers and writers allows I/O operations to be performed in parallel, instead of sequentially.

The performance of a parallel backup is limited by the slowest component in the system. It is typically a physical disk, but it could also be other components, such as the I/O controller or the system bus. Each of these components has a maximum rate at which they can transfer data.

The BACKUP DATABASE statement and the Backup utility (dbbackup) provide options that let you configure the behavior of a parallel backup, including:

- when and how the checkpoint log is copied
- the maximum number of pages used at a time to transfer data from the database server to dbbackup (only available when using dbbackup)
- adding more writers (BACKUP statement only)

Backups should always be made to a separate physical drive. This provides a performance benefit from the I/O parallelism, and also improves the safety of the data in the event of a hardware failure.

See also

- "BACKUP statement" [*SQL Anywhere Server - SQL Reference*]
- "Backup utility (dbbackup)" on page 767

Validating databases

Database file corruption may not be confirmed until the database server tries to access the affected part of the database. You should periodically check that your database is valid by using tools such as the **Validate Database Wizard** in Sybase Central, or the Validation utility (dbvalid). You must have VALIDATE authority to perform validation activities. See "VALIDATE authority" on page 447.

Depending on the options you specify, validation can include checksums, correctness of index data, and whether all table pages belong to objects in the database. Express database validation (the -fx option) does not validate data, continued row structure, or foreign key relationships.

Validation requires exclusive access to the object being validated. For this reason, it is best to validate when there is no other activity on the database. If you can be sure that no transactions are in progress when the backup is being made, the database server does not need to perform recovery steps. In this case, you can perform a validity check on the backup using the read-only database option. See "-r dbeng12/dbsrv12 server option" on page 217.

Tip
Using the BACKUP statement with the WAIT BEFORE START clause ensures that no transactions are in progress when you start a backup.

If a base table in the database file is corrupt, you should treat the situation as a media failure, and recover from your previous backup. If an index is corrupt, you may want to unload the database without indexes, and reload.

See also

- "Validate a database" on page 930
- "Validating the transaction log" on page 922
- "Validate a table" on page 931
- "VALIDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "Improving performance when validating databases" on page 932

Using checksums to detect corruption

Checksums are used to determine whether a database page has been modified on disk. SQL Anywhere supports two types of checksums: global checksums and write checksums.

Global checksums

Global checksums are enabled when a database is created. When you create a database with global checksums enabled, a checksum is calculated for each page just before it is written to disk. The next time the page is read from disk, the page's checksum is recalculated and compared to the checksum stored on the page. If the checksums are different, then the page has been modified on disk and an error occurs.

You can check whether a database was created with global checksums enabled by executing the following statement:

```
SELECT DB_PROPERTY ( 'Checksum' );
```

This query returns ON if global checksums are turned on; otherwise, it returns OFF.

The CHECKSUM clause of the ALTER DATABASE statement lets you disable global checksums for an existing database. Disabling checksums is not recommended. Once global checksums are disabled, they cannot be enabled. You can either use the -wc+ option to enable write checksums for the database, or you can rebuild the database with checksums enabled.

Write checksums

By default, databases created with version 10 and 11 of SQL Anywhere do not have global checksums enabled. If you start a database created with SQL Anywhere 11 on a version 12 database server, then by default the database server creates write checksums, which are checksums that are added to pages only when they are written to disk.

Version 12 databases have global checksums enabled by default. If you create a new database and disable global checksums for the database, you can enable write checksums by using the -wc+ option or the START DATABASE statement.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can check whether a database was started with write checksums by executing the following statement:

```
SELECT DB_PROPERTY ( 'WriteChecksum' );
```

This query returns ON if checksums are enabled only when pages are written to disk (because global checksums or write checksums are enabled); otherwise, it returns OFF.

See:

- "-wc dbeng12/dbsrv12 server option" on page 235
- "-wc dbeng12/dbsrv12 database option" on page 262
- "START DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "WriteChecksum database property" on page 675

Automatic write checksum creation

In the following situations, write checksums are enabled for the database, regardless of the global checksum setting that was specified when the database was created:

- **Critical pages** The database server calculates write checksums for critical database pages in all databases, regardless of whether global checksums are enabled. These checksums are used to detect offline corruption, which can help reduce the chances of other data being corrupted as the result of a bad critical page. Because the database server calculates these checksums, if a database becomes corrupt that does not have checksums enabled, the database server shuts down with a fatal error.

  As well, if you validate a database that does not have global checksums enabled, but that has a bad critical page, dbvalid can still return warnings about checksum violations.

- **Windows Mobile databases** The database server automatically enables write checksums for databases running on Windows Mobile to help provide early detection if the database file becomes corrupt.

- **Databases running on some storage media** When the database is running on storage media that may be less reliable, such as network or removable drives, the database server automatically enables write checksums for the database. Write checksums remain enabled as long as the database resides on such a device, and the pages are checksummed when they are written. If the database is moved to a more reliable storage device, the database server verifies the checksum for checksummed pages when they are brought into the database server cache.

See:

- "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "Checksum database property" on page 662

Validating checksums

You can validate disk pages for databases that use global or write checksums. If a database has global checksums enabled, then all database pages are validated. If a database has used only write checksums, then only pages with checksums are validated. Checksum validation requires either DBA or VALIDATE authority.

For databases with checksums enabled, a checksum is calculated for each database page and this value is stored when the page is written to disk. You can use the Validation utility (dbvalid), the VALIDATE statement, the sa_validate system procedure, or the **Validate Database Wizard** in Sybase Central to perform checksum validation, which consists of reading the database pages from disk and calculating the checksum for the page. If the calculated checksum does not match the stored checksum for a page, the page has been modified or corrupted while on disk or while writing to the page. If one or more pages has been corrupted, an error is returned and information about the invalid pages appears in the database server messages window.

See:

- "VALIDATE statement" [*SQL Anywhere Server - SQL Reference*]
- "Validation utility (dbvalid)" on page 881
- "sa_validate system procedure" [*SQL Anywhere Server - SQL Reference*]

Validate a database

You must have either DBA or VALIDATE authority to validate a database.

Caution
Validating a table or an entire database should be performed while no connections are making changes to the database; otherwise, errors may be reported indicating some form of database corruption even though no corruption actually exists.

To check the validity of an entire database (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, select the database.
3. From the **File** menu, choose **Validate Database**.
4. Follow the instructions in the **Validate Database Wizard**.

Tip
You can also access the **Validate Database Wizard** from within Sybase Central using any of the following methods:

- Right-clicking the database, and choosing **Validate Database**.
- Selecting the database, and choosing **Tools** » **SQL Anywhere 12** » **Validate Database**.

To check the validity of an entire database (SQL)

- Execute the sa_validate stored procedure:

```
CALL sa_validate;
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The procedure returns a single column, named Messages. If all tables are valid, the column contains `No errors detected`.

For more information, see "sa_validate system procedure" [*SQL Anywhere Server - SQL Reference*].

To check the validity of an entire database (command line)

- Run the dbvalid utility:

```
dbvalid -c "connection-string"
```

See "Validation utility (dbvalid)" on page 881.

Note
If you are checking the validity of a backup copy, run the database in read-only mode so that it is not modified in any way. You can only do this if there were no transactions in progress during the backup. See "-r dbeng12/dbsrv12 server option" on page 217.

Validate a table

You must have either DBA or VALIDATE authority to validate a table.

To check the validity of a table (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, double-click **Tables**.
3. Right-click the table and choose **Validate**.
4. Click **OK**.

To check the validity of a table (SQL)

- Execute the VALIDATE TABLE statement:

```
VALIDATE TABLE table-name;
```

Notes

- If errors are reported, you can drop all the indexes and keys on a table and recreate them. Any foreign keys to the table also need to be recreated.
- If you suspect a particular index, you can execute an ALTER INDEX ... REBUILD statement to rebuild the corrupted index. See "ALTER INDEX statement" [*SQL Anywhere Server - SQL Reference*].

- Another solution for errors reported by VALIDATE TABLE is to unload and reload your entire database. You should use the dbunload -u option so that the unload process does not try to use a possibly corrupt index to order the data.

Improving performance when validating databases

The VALIDATE TABLE statement can be slow when used on large databases running on servers with a cache size too small to contain the table and its largest index. It is often the case that all pages in the table are read at least once for each index. As well, if full compares are required for index lookups, the number of page reads can be proportional to the number of rows (not pages) in the table.

If you want to reduce the time taken to validate, you can use the WITH EXPRESS CHECK option with the VALIDATE TABLE statement, or the -fx option with the dbvalid utility. Depending on the size of your database, the size of your cache, and the type of validation you require, these two features can significantly reduce the time taken to perform validation.

Express validation causes each row of the table to be read and all columns evaluated. Each index is completely scanned once, and checks are done to ensure that the rows referenced in the index exist in the table. The express check option also does checks on the validity of individual index pages. The number of rows in the table must match the number of entries in the index. The express option saves time because it does not perform individual index lookups for each row.

Because the express check feature does not perform individual lookups, it is possible (though unlikely) for some form of index corruption to go unnoticed by the express validation feature. If index corruption should occur, data can be recovered by unloading and rebuilding the database since validation has confirmed that all the data can be read. You can also use the REBUILD clause of the ALTER INDEX statement to correct index corruption. See “ALTER INDEX statement” [*SQL Anywhere Server - SQL Reference*].

- “VALIDATE statement” [*SQL Anywhere Server - SQL Reference*]
- “Validation utility (dbvalid)” on page 881
- “sa_validate system procedure” [*SQL Anywhere Server - SQL Reference*]

Automating tasks using schedules and events

Many database administration tasks are best performed systematically. For example, a regular backup procedure is an important part of proper database administration procedures.

You can automate routine tasks in SQL Anywhere by adding an **event** to a database, and providing a schedule for the event. Whenever one of the times in the schedule passes, the database server executes a sequence of actions called an **event handler**.

Database administration also requires taking action when certain conditions occur. For example, it may be appropriate to email a notification to a system administrator when a disk containing the transaction log is filling up so that the administrator can handle the situation. These tasks too can be automated by defining event handlers for one of a set of **system events**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Questions and answers

To answer the question ...	Consider reading ...
What is a schedule?	“Understanding schedules” on page 934
What is an event?	“Understanding events” on page 933
What is a system event?	“Understanding system events” on page 935
What is an event handler?	“Understanding event handlers” on page 939
How do I debug event handlers?	“Developing event handlers” on page 939
How does the database server use schedules to trigger event handlers?	“How the database server checks for scheduled events” on page 940
How can I schedule regular backups?	“Understanding schedules” on page 934
What kind of system events can the database server use to trigger event handlers?	“Understanding system events” on page 935 “CREATE EVENT statement” [*SQL Anywhere Server - SQL Reference*]
What connection do event handlers get executed on?	“How event handlers are executed” on page 941
How do event handlers get information about what triggered them?	“Developing event handlers” on page 939 “EVENT_PARAMETER function [System]” [*SQL Anywhere Server - SQL Reference*]

Understanding events

You can automate routine tasks in SQL Anywhere by adding an event to a database, and providing a schedule for the event. SQL Anywhere supports three types of events:

- **Scheduled events** have an associated schedule and execute at specified times. See “Understanding schedules” on page 934.
- **System events** are associated with a particular type of condition that is tracked by the database server. See “Understanding system events” on page 935.
- **Manual events** are fired explicitly using the TRIGGER EVENT statement. See “Triggering an event handler” on page 943.

After each execution of an event handler, a COMMIT occurs if no errors occurred. A ROLLBACK occurs if there was an error.

Understanding schedules

By scheduling activities you can ensure that a set of actions is executed at a set of preset times. The scheduling information and the event handler are both stored in the database itself.

Although this is not usually necessary, you can define complex schedules by associating more than one schedule with a named event. For example, a retail outlet might want an event to occur once per hour during hours of operation, where the hours of operation vary based on the day of the week. You can achieve the same effect by defining multiple events, each with its own schedule, and by calling a common stored procedure.

When scheduling events, you can use either full-length English day names (Monday, Tuesday, and so on) or the abbreviated forms of the day (Mon, Tue, and so on). Note that you must use the full-length English day names if you want the day names to be recognized by a server running in a language other than English.

Events are not fired on a server that is currently acting as the mirror server. As well, mirroring events cannot be defined to execute on an arbiter, since events only run in the context of the database in which they are defined, and the arbiter does not use a copy of the database being mirrored.

The following examples give some ideas for scheduled actions that may be useful.

Examples

Perform an incremental backup daily at 1:00 A.M.:

```
CREATE EVENT IncrementalBackup
SCHEDULE
 START TIME '1:00 AM' EVERY 24 HOURS
HANDLER
BEGIN
 BACKUP DATABASE DIRECTORY 'c:\\backup'
 TRANSACTION LOG ONLY
 TRANSACTION LOG RENAME MATCH
END;
```

Summarize orders at the end of each business day:

```
CREATE EVENT Summarize
SCHEDULE
 START TIME '6:00 pm'
 ON ( 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
  'Friday' )
HANDLER
 BEGIN
 INSERT INTO OrderSummary
  SELECT CURRENT DATE,
     COUNT( * ),
     SUM( amount )
  FROM Orders
  WHERE date_ordered = current date
END;
```

See also

- "CREATE EVENT statement" [*SQL Anywhere Server - SQL Reference*]

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Defining schedules

To permit flexibility, schedule definitions have several components to them:

- **Name** Each schedule definition has a name. You can assign more than one schedule to a particular event, which can be useful in designing complex schedules.
- **Start time** You can define a start time for the event, which is the time when execution begins.
- **Range** As an alternative to a start time, you can specify a range of times for which the event is active. The event occurs between the start and end time specified. Frequency is determined by the specified recurrence.
- **Recurrence** Each schedule can recur. The event is triggered on a frequency that can be given in hours, minutes, or seconds on a set of days that can be specified as days of the week or days of the month. Recurring events include an **EVERY** or **ON** clause.

You can define the schedule for an event in the CREATE EVENT statement, or using the Create Schedule Wizard.

For information about adding a schedule when creating an event, see "CREATE EVENT statement" [*SQL Anywhere Server - SQL Reference*].

To create a schedule for an event (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to your database as a user with DBA authority.
2. Double-click **Events**.
3. Double-click the event for which you want to create a schedule.
4. Click the **Schedules** tab.
5. From the **File** menu, choose **New** » **Schedule**.
6. Follow the instructions in the **Create Schedule Wizard**.

Understanding system events

SQL Anywhere tracks several system events. Each system event provides a hook on which you can hang a set of actions. The database server tracks the events for you, and executes the actions (as defined in the event handler) when the system event satisfies a provided **trigger condition**.

For more information about trigger conditions, see "Defining trigger conditions for events" on page 937.

By defining event handlers to execute when a chosen system event occurs and satisfies a trigger condition that you define, you can improve the security and safety of your data, and help ease administration. The actions of an event handler are committed if no error is detected during execution, and rolled back if errors are detected.

The available system events include the following:

- **BackupEnd** You can use the BackupEnd event type to take action at the end of a backup.
- **Connection events** When a connection is made (Connect) or when a connection attempt fails (ConnectFailed). You may want to use these events for security purposes. As an alternative to a connect event handler, you may want to consider using a login procedure. See "login_procedure option" on page 549.
- **DatabaseStart** You can use the DatabaseStart event type to take action when a database is started.
- **Deadlock** You can use the Deadlock event to take action when a deadlock occurs. The event handler can use the sa_report_deadlocks procedure to obtain information about the conditions that led to the deadlock. When using the Deadlock event, you should configure the database server to capture deadlock information by setting the log_deadlocks option to On, and by enabling the RememberLastStatement feature using sa_server_option or the -zl server option.

  Deadlock events fire for connection deadlocks and thread deadlocks. A deadlock event provides no information beyond what is available via the sa_report_deadlocks system procedure. However, using this event allows you to act on the deadlock in a timely manner. A quick response may be important since the amount of deadlock-related information the database server maintains is limited. See:

  - "sa_report_deadlocks system procedure" [*SQL Anywhere Server - SQL Reference*]
  - "log_deadlocks option" on page 547
  - "Deadlock" [*SQL Anywhere Server - SQL Usage*]
- **Disconnect** You can use the Disconnect event to take action when a user or application disconnects.
- **Free disk space** Tracks the available disk space on the device holding the database file (DBDiskSpace), the log file (LogDiskSpace), or temporary file (TempDiskSpace). This system event is not available on Windows Mobile.

  You may want to use disk space events to alert administrators of a disk space shortage.

  You can specify the -fc option when starting the database server to implement a callback function when the database server encounters a file system full condition. See "-fc dbeng12/dbsrv12 server option" on page 181.
- **File size** The file reaches a specified size. This can be used for the database file (GrowDB), the transaction log (GrowLog), or the temporary file (GrowTemp).

  You may want to use file size events to track unusual actions on the database, or monitor bulk operations.
- **GlobalAutoincrement** When the number of remaining values for a column defined with GLOBAL AUTOINCREMENT is less than one percent of its range, the GlobalAutoincrement event fires. This can be used to request a new value for the global_database_id option based on the table and number of remaining values that are supplied as parameters to this event. To get the remaining values for the table within the event, use the EVENT_PARAMETER function with the RemainingValues and TableName parameters. RemainingValues returns the number of remaining values that can be generated for the column, while TableName returns the table containing the GLOBAL

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

AUTOINCREMENT column that is near the end of its range. See "EVENT_PARAMETER function [System]" [*SQL Anywhere Server - SQL Reference*].

- **RAISERROR error** When a RAISERROR statement is executed, you can use the RAISERROR event type to take actions. The error number used in the RAISERROR statement can be determined within the event handler using the EVENT_CONDITION function (for example, `EVENT_CONDITION( 'ErrorNumber' )`).

- **Idle time** The database server has been idle for a specified time (ServerIdle). You may want to use this event type to perform routine maintenance operations at quiet times.

- **Database mirroring** When the connection from the primary server to a mirror server or arbiter server is lost, the MirrorServerDisconnect event fires. To get the name of the server whose connection was lost, use the EVENT_PARAMETER function with the MirrorServerName parameter. See "EVENT_PARAMETER function [System]" [*SQL Anywhere Server - SQL Reference*].

  The MirrorFailover event fires whenever a server takes ownership of the database. For example, it fires when a server first starts and determines that it should own the database. It also fires when a server previously acting as the mirror determines that the primary server has gone down and, after consulting with the arbiter, determines that it should take ownership.

  Events are not fired on a server that is currently acting as the mirror server since its copy of the database is still being started. As well, mirroring events cannot be defined to execute on an arbiter, since events only run in the context of the database in which they are defined, and the arbiter does not use a copy of the database being mirrored. See "Database mirroring system events" on page 971.

Defining trigger conditions for events

Each event definition has a system event associated with it. It also has one or more trigger conditions. The event handler is triggered when the trigger conditions for the system event are satisfied.

The trigger conditions are included in the WHERE clause of the CREATE EVENT statement, and can be combined using the AND keyword. Each trigger condition is of the following form:

event_condition(*condition-name*) *comparison-operator value*

The *condition-name* argument is one of a set of preset strings, which are appropriate for different event types. For example, you can use **DBSize** (the database file size in megabytes) to build a trigger condition suitable for the **GrowDB** system event. The database server does not check that the condition-name matches the event type: it is your responsibility to ensure that the condition is meaningful in the context of the event type.

Examples

- Limit the transaction log size to 10 MB:

```
CREATE EVENT LogLimit
TYPE GrowLog
WHERE event_condition( 'LogSize' ) > 10
HANDLER
```

```
BEGIN
  IF EVENT_PARAMETER( 'NumActive' ) = 1 THEN
   BACKUP DATABASE
   DIRECTORY 'c:\\logs'
   TRANSACTION LOG ONLY
   TRANSACTION LOG RENAME MATCH;
  END IF;
END;
```

- Notify an administrator when free disk space on the device containing the database file falls below 10%, but do not execute the handler more than once every five minutes (300 seconds):

```
CREATE EVENT LowDBSpace
TYPE DBDiskSpace
WHERE event_condition( 'DBFreePercent' ) < 10
AND event_condition( 'Interval' ) >= 300
HANDLER
BEGIN
 CALL xp_sendmail( recipient='DBAdmin',
    subject='Low disk space',
    "message"='Database free disk space '
    || EVENT_PARAMETER( 'DBFreeSpace' ) );
END;
```

- Notify an administrator of a possible attempt to break into the database:

```
CREATE EVENT SecurityCheck
TYPE ConnectFailed
HANDLER
BEGIN
 DECLARE num_failures INT;
 DECLARE mins INT;
 INSERT INTO FailedConnections( log_time )
 VALUES ( CURRENT TIMESTAMP );

 SELECT COUNT( * ) INTO num_failures
 FROM FailedConnections
 WHERE log_time >= DATEADD( minute, -5,
  current timestamp );
 IF( num_failures >= 3 ) THEN
  SELECT DATEDIFF( minute, last_notification,
   current timestamp ) INTO mins
  FROM Notification;
  IF( mins > 30 ) THEN
   UPDATE Notification
   SET last_notification = current timestamp;
   CALL xp_sendmail( recipient='DBAdmin',
      subject='Security Check', "message"=
       'over 3 failed connections in last 5 minutes' )
  END IF
 END IF
END;
```

- Run a process when the server has been idle for ten minutes. Do not execute more frequently than once per hour:

```
CREATE EVENT Soak
TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) >= 600
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
AND event_condition( 'Interval' ) >= 3600
HANDLER
BEGIN
 MESSAGE ' Insert your code here ... '
END;
```

Understanding event handlers

Event handlers execute on a separate connection from the action that triggered the event, and so do not interact with client applications. They execute with the permissions of the creator of the event.

Developing event handlers

Event handlers, whether for scheduled events or for system event handling, contain compound statements, and are similar in many ways to stored procedures. You can add loops, conditional execution, and so on, and you can use the SQL Anywhere debugger to debug event handlers.

After each execution of an event handler, a COMMIT occurs if no errors occurred. A ROLLBACK occurs if there was an error.

Context information for event handlers

Unlike stored procedures, event handlers do not take any arguments. You can use the EVENT_PARAMETER function to access information about the context in which an event was triggered. The information returned includes the connection ID and user ID that caused an event to be triggered, and the event name and the number of times it has been executed. See "EVENT_PARAMETER function [System]" [*SQL Anywhere Server - SQL Reference*].

Testing event handlers

During development, you want event handlers to be triggered at convenient times. You can use the TRIGGER EVENT statement to explicitly cause an event to execute, even when the trigger condition or scheduled time has not occurred. However, TRIGGER EVENT does not cause disabled event handlers to be executed. See "TRIGGER EVENT statement" [*SQL Anywhere Server - SQL Reference*].

While it is not good practice to develop event handlers on a production database, you can disable event handlers from Sybase Central or explicitly using the ALTER EVENT statement.

Code sharing

It can be useful to use a single set of actions to handle multiple events. For example, you may want to take a notification action if disk space is limited on any of the devices holding the database or log files. To do this, create a stored procedure and call it in the body of each event handler, passing any needed context information as parameters to the procedure.

Debugging event handlers

Debugging event handlers is very similar to debugging stored procedures. The event handlers appear in the events list.

For more information and step-by-step instructions, see "Debugging an event handler" on page 944.

Hiding event handlers

You can use the SET HIDDEN clause to hide the definition of an event handler. Specifying the SET HIDDEN clause results in the permanent obfuscation of the event handler definition stored in the action column of the ISYSEVENT system table. See "ALTER EVENT statement" [*SQL Anywhere Server - SQL Reference*].

Limiting active events

You can also determine how many instances of a particular event handler are currently active using the NumActive event parameter. This function is useful if you want to limit an event handler so that only one instance executes at any given time.

For more information about the NumActive event parameter, see "EVENT_PARAMETER function [System]" [*SQL Anywhere Server - SQL Reference*].

Schedule and event internals

This section describes how the database server processes schedules and event definitions.

How the database server checks for system events

System events are classified according to their **event type**, as specified directly in the CREATE EVENT statement or using Sybase Central. There are two kinds of event types:

- **Active event types** Some event types are the result of action by the database server itself. These active event types include growing database files, or the start and end of different database actions (BackupEnd and so on) or RAISERROR.

  When the database server takes the action, it checks to see whether the trigger conditions defined in the WHERE clause are satisfied, and if so, triggers any events defined for that event type.

- **Polled event types** Some event types, such as free disk space types (DBDiskSpace and so on) and IdleTime type, are not triggered solely by database actions.

  For these types of events, the database server polls every thirty seconds, starting approximately thirty seconds after the database server is started.

  For the IdleTime event type, the database server checks whether the server has been idle for the entire thirty seconds. If no requests have started and none are currently active, it adds the idle check interval time in seconds to the idle time total; otherwise, the idle time total is reset to 0. The value for IdleTime is therefore always a multiple of thirty seconds. When IdleTime is greater than the interval specified in the trigger condition, event handlers associated with IdleTime are fired.

How the database server checks for scheduled events

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The calculation of scheduled event times is done when the database server starts, and each time a scheduled event handler completes.

The calculation of the next scheduled time is based on the increment specified in the schedule definition, with the increment being added to the previous start time. If the event handler takes longer to execute than the specified increment, so that the next time is earlier than the current time, the database server increments until the next scheduled time is in the future.

For example, an event handler that takes sixty-five minutes to execute and is requested to run every hour between 9:00 and 5:00 will run every two hours, at 9:00, 11:00, 1:00, and so on.

To run a process such that it operates between 9:00 and 5:00 and delays for some period before the next execution, you could define a handler to loop until its completion time has passed, with a WAITFOR statement between each iteration.

If you are running a database server intermittently, and it is not running at a scheduled time, the event handler does not run at startup. Instead, the next scheduled time is computed at startup. If, for example, you schedule a backup to take place every night at one o'clock, but regularly shut down the database server at the end of each work day, the backup never takes place.

If the next scheduled execution of an event is more than one hour away, the database server will recalculate its next scheduled time on an hourly basis. This allows events to fire when expected when the system clock is adjusted because of a change to or from Daylight Savings Time.

How event handlers are executed

When an event handler is triggered, a temporary internal connection is made on which the event handler is executed. The handler is not executed on the connection that caused the handler to be triggered, so statements such as MESSAGE ... TO CLIENT, which interact with the client application, are not meaningful within event handlers. Similarly, statements that return result sets are not permitted.

The temporary connection on which the handler is executed does not count towards the connection limit for licensing purposes, and the procedure specified by the login_procedure option is not executed for event connections.

Event creation requires DBA authority, and events execute with the permissions of their creator. If you want event handlers to execute with non-DBA authority, you can call a procedure from within the handler, as stored procedures run with the permissions of their creator.

Any event errors are logged to the database server message log.

Event handlers and errors

The transaction in an event handler is committed if no errors are detected during execution, and rolled back if errors are detected.

If an error occurs within an atomic compound statement and that statement has an exception handler that handles the error, then any changes made within the statement are left outstanding. If the exception handler does not handle the error or causes another error (including via RESIGNAL), then changes made within the atomic statement are undone.

Event handling tasks

This section collects together instructions for tasks related to automating tasks with events.

Adding an event to a database

You can add events from Sybase Central and by using SQL.

To add an event to a database (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, right-click **Events** and choose **New** » **Event**.
3. Follow the instructions in the **Create Event Wizard**.

   Detailed explanations for event options are explained in other tasks.

To add an event to a database (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a CREATE EVENT statement.

   The CREATE EVENT statement contains many options, depending on the event you want to create. These are explained in detail in other tasks.

   See "CREATE EVENT statement" [*SQL Anywhere Server - SQL Reference*].

Adding a manually-triggered event to a database

If you create an event handler without a schedule or system event to trigger it, it is executed only when manually triggered.

To add a manually-triggered event to a database (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. In the left pane, right-click **Events** and choose **New** » **Event**.
3. In the **What Do You Want To Name The New Event** field, type a name for the event and click **Next**.
4. Select **Manually** and click **Next**.
5. Select **Enable This Event**, **Execute At All Databases**, and then click **Next**.
6. Type a comment describing the event and click **Finish**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

7. In the **SQL** pane, type the SQL statements for your event.

8. From the **File** menu, choose **Save**.

To add a manually-triggered event to a database (SQL)

1. Connect to the database as a user with DBA authority.

2. Execute a CREATE EVENT statement with no schedule or WHERE clause. The restricted syntax of the CREATE EVENT is as follows:

   CREATE EVENT *event-name*
   HANDLER
   BEGIN
   ... //event handler
   END

If you are developing event handlers, you can add schedules or system events to control the triggering of an event later, either using Sybase Central or the ALTER EVENT statement.

See also

- "Triggering an event handler" on page 943
- "ALTER EVENT statement" [*SQL Anywhere Server - SQL Reference*]

Triggering an event handler

Any event handler can be triggered manually, in addition to those occasions when it executes because of a schedule or system event. Triggering events manually can be useful during development of event handlers, and also, for certain events, in production environments. For example, if you have a monthly sales report scheduled, you might want to obtain a sales report for a reason other than the end of the month.

For more information about developing event handlers, see "Developing event handlers" on page 939.

To trigger an event handler (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.

2. In the left pane, double-click **Events**.

3. Right-click the event and choose **Trigger**.

   The event must be enabled before you can trigger it. To enable an event, right-click it and choose **Enabled**.

4. In the **Parameters** field, type a comma-separated list of parameters for the event. For example:

   ```
   parameter=value,parameter=value
   ```

5. Click **OK**.

To trigger an event handler (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute the TRIGGER EVENT statement, supplying the name of the event. For example:

   ```
   TRIGGER EVENT sales_report_event;
   ```

   See "TRIGGER EVENT statement" [*SQL Anywhere Server - SQL Reference*].

Debugging an event handler

Debugging is a regular part of any software development. Event handlers can be debugged during the development process.

To debug an event handler (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.

   Connect to the database as a user with DBA authority.
2. From the **Mode** menu, choose **Debug**.
3. In the left pane, double-click **Events**.
4. Double-click the event you want to debug.
5. On the **SQL** tab in the right pane, press F9 to set a breakpoint.
6. From Interactive SQL or another application, trigger the event handler using the TRIGGER EVENT statement.
7. The execution stops at the breakpoint you set.

See also

- "Developing event handlers" on page 939
- "Debugging procedures, functions, triggers, and events" [*SQL Anywhere Server - SQL Usage*]

Hiding an event handler

For improved security, you can hide the definition for an event handler using the ALTER EVENT statement. This results in the obfuscation of the event handler definition stored in the action column of the ISYSEVENT system table.

To hide an event handler (SQL)

1. Connect to the database as a user with DBA authority.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

2. Execute the ALTER EVENT *event-name* SET HIDDEN statement, where *event-name* is the name of the event for which you are hiding the handler.

See also

- "ALTER EVENT statement" [*SQL Anywhere Server - SQL Reference*]
- "SYSEVENT system view" [*SQL Anywhere Server - SQL Reference*]

SQL Anywhere high availability

SQL Anywhere supports database mirroring and Veritas Cluster Server to provide high availability. Both of these configurations provide alternate database servers that can run a database should the database server currently running the database become unavailable.

If you want to distribute workloads, then you should also consider whether using read-only scale-out is an appropriate solution. Read-only scale-out can be used with database mirroring. See "SQL Anywhere read-only scale-out" on page 980 and "Using read-only scale-out with database mirroring" on page 989.

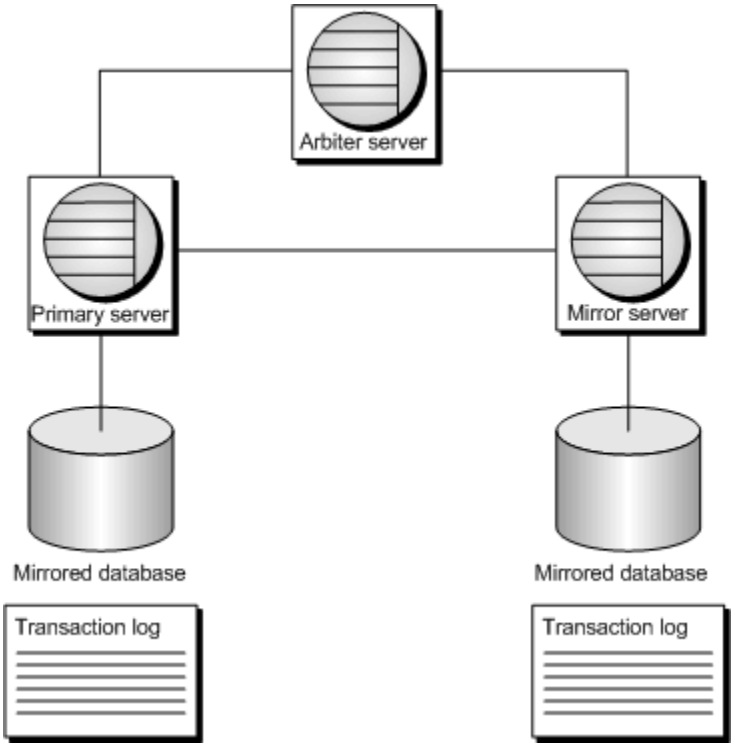
For information about using high availability and/or read-only scale-out for hosting web applications, see "SQL Anywhere web services high availability and scale-out solutions" on page 990.

Introduction to database mirroring

Separately licensed component required
Database mirroring requires a separate license. See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

Database mirroring is a configuration of either two or three database servers, running on separate computers, that co-operate to maintain copies of the database and transaction log files.

The **primary server** and **mirror server** each maintain a copy of the database files and transaction log files, while the third server, called the **arbiter server**, is used when it is necessary to determine which of the other two servers can take ownership of the database. The arbiter does not maintain a copy of the database. The configuration of three database servers (the primary, mirror, and arbiter servers) is called a **mirroring system**, and the primary and mirror servers together are called the **partner** servers.



Clients connect to the primary server to access the database. Any changes that are made to the database are recorded in the transaction log on the primary server. When the changes are committed, the transaction log pages are sent to the mirror server where they are applied to a mirror copy of the database. The copy of the database on the mirror server can only be accessed in read-only mode while that server is acting as the mirror server. See "Configuring read-only access to a database running on the mirror server" on page 967.

If the primary server becomes unavailable because of hardware or software failure, the mirror server negotiates with the arbiter to take ownership of the database and assume the role of primary server. For an ownership transfer, or **role switch**, to take place, the surviving partner server and the arbiter must agree that the mirror was in a current, synchronized state at the time the role switch is attempted. Any clients that were connected to the original primary server are disconnected, and any uncommitted transactions are lost. Clients must then reconnect to the database on the new primary server to continue accessing the database. When the original primary server becomes available again, it assumes the role of mirror server.

The database servers display status messages in the database server messages window on startup to indicate which role the server is assuming and how far the startup process has progressed. A message appears if the database must be restarted because of the loss of one or more of the other servers in the mirroring system, or if its role changes from mirror to primary.

If an assertion failure occurs on a server that is part of a mirroring system, the server writes the error to the database server message log and then exits. This notifies the other servers that it has failed so that they can take appropriate action.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

There are no special hardware or software requirements for database mirroring, and the database servers can be running in separate geographical locations. Database servers that are participating in a database mirroring system can run both mirrored and non-mirrored databases. As well, the arbiter server can be the arbiter for multiple database mirroring systems.

Details about the state of each database in the database mirroring system are stored in a state information file. See "State information files" on page 952.

Note
Database mirroring is not a replacement for a backup and recovery plan. You should always implement a backup and recovery strategy for your database. See "Database mirroring and backups" on page 972 and "Backup and data recovery" on page 887.

For information about upgrading SQL Anywhere or rebuilding a database involved in a database mirroring system, see "Upgrading SQL Anywhere software and databases in a database mirroring system" [*SQL Anywhere 12 - Changes and Upgrading*].

Quorum

Before a server can assume the role of primary server, it must have a **quorum**, which means that at least one other server must agree that a server can own the database. If the mirror server becomes unavailable while the primary server and arbiter are connected, the primary server continues to provide access to the database. If the primary server loses quorum, it can no longer permit access to the database. At that point, it stops the mirrored database, attempts to restart it, and then waits to regain quorum before making the database available.

When you start a database mirroring system, the database servers go through a startup process to reach quorum and accept client connections. The following steps describe a typical sequence of events for this process:

1. The arbiter server waits for Server 1 and Server 2.
2. Server 1 looks for the arbiter server or Server 2.
3. Server 1 connects to the arbiter server.
4. Server 1 negotiates with the arbiter server to become the primary server.
5. The arbiter server and Server 1 agree that Server 1 can become the primary server.
6. Server 1 starts accepting connections.
7. Server 2 looks for Server1 and the arbiter.
8. Server 2 connects to the arbiter and to Server 1.
9. Server 2 requests quorum. It does not receive quorum because Server 1 is the primary, and so it stands by waiting for transactions from Server 1.
10. Server 1 sends transactions to Server 2.

Restrictions

The following restrictions apply when using database mirroring:

- **Network database server required** Because mirroring involves network communication between the database servers, you must use the network database server (dbsrv12); the personal database server cannot be used.
- **LOAD TABLE statement** If you are using database mirroring and execute a LOAD TABLE statement on a base table, you must specify either WITH ROW LOGGING or WITH CONTENT LOGGING as the logging level for the statement. These clauses allow the loaded data to be recorded in the transaction log so that it can be loaded into the mirroring database as well. If these clauses are not specified, an error is reported. See "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*] and "Import data with the LOAD TABLE statement" [*SQL Anywhere Server - SQL Usage*].
- **TCP/IP required** Only TCP/IP connections are permitted between mirroring servers.
- **Failover and scheduled events** If your database has scheduled events, and failover occurs, the failover must complete before the events start; otherwise, the events are not executed until their next scheduled time. If the mirror server is changing to the primary server role but has not completed the transition, the scheduled event is not executed; it is executed at the next scheduled time. When an event is running and the primary server loses its connections to the mirror and arbiter servers, the event connection and all other connections are dropped. If an event is scheduled to run after the mirror assumes the primary server role, the event runs on the new primary server.
- **Transaction log restrictions** You cannot truncate the transaction log when you are using database mirroring because this may result in lost transactions. You can rename the transaction log as often as necessary. If you want to remove old transaction logs, you can use a scheduled event to delete them once you are certain that they are no longer needed. For example, you could create an event that runs each day and deletes copies of the transaction log that are more than a week old. See "Database mirroring and transaction log files" on page 971.
- **Using SQL Anywhere web servers in a mirroring system** If you are using SQL Anywhere as a web server and in a mirroring system, it is not possible to specify a URL for a web request in a way that guarantees that the request will be directed to the current primary server. If one of the server computers is named in the URL and that server is down, the request times out.

Considerations when developing applications

When you are using database mirroring, in almost all cases, applications should be able to run in the same manner as they do when connected to a non-mirrored database. However, there are a few considerations to take into account when developing applications that are used with database mirroring:

- Create clients that can reconnect to the database (for example, when failover occurs the user may need to shut down the application and then restart it).
- When running in asynchronous or asyncfullpage mode, you must determine what happens when failover occurs and transactions are not committed to the database.
- Incomplete transactions must be rolled back when the mirror server takes ownership of the database, and the longer a transaction is, the longer it takes to roll the transaction back. The recovery speed for

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

failover is affected by the number of clients and the length of their transactions that need to be rolled back. If recovery speed is a concern, you may want to design your application to use short transactions whenever possible.

Upgrading SQL Anywhere

For information about upgrading SQL Anywhere for a database mirroring system, including applying EBFs, see "Upgrading SQL Anywhere software and databases in a database mirroring system" [*SQL Anywhere 12 - Changes and Upgrading*].

Benefits of database mirroring

Mirroring offers several benefits:

- When an arbiter is present, failover from primary to mirror is automatic. If you are running in synchronous mode, no committed transactions are lost during failover.
- Failover is very fast because the mirror server has already applied the transaction log. When the mirror detects that the primary has failed, it rolls back any uncommitted transactions and then makes the database available.
- No special hardware, such as a shared disk is required.
- No special software (for clustering, for example) is required.
- No particular operating system version is required.
- The servers do not need to be located near each other geographically. In fact, locating them far apart provides additional protection against disasters such as fire.
- Database servers in a mirroring system can also be used to run other databases.

Understanding the role of the arbiter server

The arbiter server resolves disputes between the servers regarding which server should be the primary server. Without an arbiter, if server A starts up when server B is unavailable, server A can not determine if its copy of the database files is the most current. Starting a database using files that are not current results in the loss of transactions that have already been applied and committed to the other copy of the database. In addition, the other copy of the database would be unusable for mirroring once the two partner servers re-established communication.

In addition to resolving disputes at startup, the arbiter is involved if the communication link between two servers is broken, but both of those servers are still running. Without an arbiter, both servers could assume that they should take ownership of a database. Again, this would result in lost transactions and incompatible databases. With an arbiter, the primary server can verify that it still owns the database and can remain available to clients. If the primary server loses communications with both the mirror and the arbiter, it must shut down and wait for either one to become available.

An arbiter server can function as arbiter for more than one mirror system. It can also act as a database server for other databases.

Choosing a database mirroring mode

Three operational modes are provided for database mirroring:

- synchronous
- asynchronous
- asyncfullpage

These modes control when and how transactions are recorded on the mirror server, and you set them by using the SET MIRROR OPTION statement to set the value of the synchronization_mode option. Synchronous mode is the default.

When choosing a synchronization mode for your database mirroring system, you must determine whether recovery speed or the state of the data is more important when failover occurs.

You can check the database mirroring mode by querying the value of the MirrorMode database property:

```
SELECT DB_PROPERTY( 'MirrorMode' );
```

Synchronous mode

In synchronous mode, committed transactions are guaranteed to be recorded on the mirror server. Should a failure occur on the primary server, no committed transactions are lost when the mirror server takes over. In this mode, the primary server sends transaction log pages to the mirror when a transaction is committed. The mirror server acknowledges that transmission when it has written those pages to its copy of the transaction log. The primary server does not reply to the application until it receives this acknowledgement.

Using synchronous mode provides **transaction safety** because the partner servers are in a synchronized state, and changes sent to the mirror must be acknowledged before the primary can proceed.

Asynchronous mode

In asynchronous mode, committed transactions are not guaranteed to be recorded on the mirror server. In this mode, the primary server sends transaction log pages to the mirror when a transaction is committed. It does not wait for an acknowledgement from the mirror before replying to the application that the COMMIT has completed. Should a failure occur on the primary server, some committed transactions may be lost when the mirror server takes over.

Asyncfullpage mode

In asyncfullpage (or page) mode, pages are not sent on COMMIT; instead, they are sent when the page is full. This reduces the amount of traffic between the two database servers and improves the performance of the primary server. If the current log page has not been sent to the mirror for the number of seconds specified by the pagetimeout parameter, it is sent even though it is not yet full. The default pagetimeout is 5 seconds. Using this mode provides a limit on how long committed transactions are exposed to being lost

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

if the primary server goes down and the mirror server takes ownership of the database. Asyncfullpage mode implies asynchronous operation, so the primary server does not wait for an acknowledgement from the mirror.

Asynchronous and asyncfullpage mode are faster than synchronous mode, but are less reliable for the above reasons. In asynchronous or asyncfullpage mode, failover from the primary server to the mirror server is not automatic because the mirror server may not have all committed transactions that were applied on the primary server. For this reason, when using one of the asynchronous modes, a mirror server, by default, cannot take ownership of a database when the primary fails. If automatic failover is desirable in this situation (despite the likelihood of lost transactions), set the auto_failover option to on using the SET MIRROR OPTION statement. Otherwise, when the failed server is restarted, it detects whether transactions were lost. If transactions were lost, it writes a message to the database server message log and shuts down the database. The current database and transaction log must then be replaced using a backup before mirroring can continue.

For information about bringing up a server after it fails in asynchronous or asyncfullpage mode, see "Recovering from primary server failure" on page 970.

Note
It is recommended that you set the auto_failover mirroring option to on if you are using asynchronous or asyncfullpage mode. Then, if the primary server goes down, the mirror server automatically takes over as the primary server.

The synchronize_mirror_on_commit database option lets you control when database changes are guaranteed to have been sent to a mirror server when running in asynchronous or asyncfullpage mode. When you set this option to On, each COMMIT causes any changes recorded in the transaction log to be sent to the mirror server, and an acknowledgement to be sent by the mirror server to the primary server once the changes are received by the mirror server. The option can be set for specific transactions using SET TEMPORARY OPTION. It may also be useful to set the option for specific applications by examining the APPINFO string in a login procedure.

SQL Anywhere supports system events that fire when failover occurs in a database mirroring system, regardless of which mode you are using. You can use these events for such tasks as notifying the administrator when failover occurs. See "Database mirroring system events" on page 971.

See also

- "synchronize_mirror_on_commit option" on page 601
- "SET OPTION statement" [*SQL Anywhere Server - SQL Reference*]
- "SET MIRROR OPTION statement" [*SQL Anywhere Server - SQL Reference*]

Synchronization states

When a mirroring system is using synchronous mode, it can be in one of two states: synchronizing or synchronized.

Once a partner server starts and determines that it will act as the mirror, it first requests any log pages from the primary server that it does not already have. This may involve copying pages from log files other

than the current active log on the primary server. As it receives these pages, the mirror applies the changes they contain to its copy of the database. Once all pages from the primary have been received, the primary and mirror are in a synchronized state. From that point onward, any changes committed on the primary must be sent to the mirror and acknowledged by the mirror.

In asynchronous and asyncfullpage mode, the mirror requests log pages as above; however, the two servers never enter a synchronized state. Once the mirror has requested all log pages available at the primary, the primary is notified that it must send any updated pages to the mirror.

State information files

Each server in the mirroring system maintains a state information file that records that server's view of the state of the mirroring system.

The state information file is used during startup when determining the role to be assumed by a server. The server's local state is compared against that of the other servers in the database mirroring system.

You must always specify a state information file for each server in the mirroring system using the state_file option of the CREATE MIRROR SERVER statement. See "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*].

The state information file contains the following information:

Field	Description
Owner	Indicates which database server is the primary server.
State	Contains the synchronization state (one of synchronizing or synchronized) to indicate whether the server is receiving log pages or is up to date. See "Synchronization states" on page 951.
Mode	Specifies the synchronization mode (one of synchronous, asynchronous, or page). See "Choosing a database mirroring mode" on page 950.
Sequence	Contains a value indicating how many times failover has occurred on the database mirroring system. The sequence number is incremented on each role switch. It helps to determine whether a server's view of the state of the mirroring system is current. See "Introduction to database mirroring" on page 945.

The following shows sample contents for a state information file:

```
[demo]
Owner=server2
State=synchronizing
Mode=asynchronous
Sequence=35
```

If a state information file does not exist, it is created automatically. State information files should only be modified by the database server.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Tutorial: Using database mirroring

This tutorial provides instructions for setting up database mirroring and how to respond to a failover. For this tutorial, all the database servers are running on the same computer. However, each database server is typically installed on a separate computer in a production environment.

If this tutorial is used with database servers running on different computers, references to localhost in the connection strings must be changed to the actual computer names.

To simulate failover in a database mirroring system

1. Create the following directories: *c:\server1*, *c:\server2*, and *c:\arbiter*.
2. Run the following command to create a database named *mirror_demo.db* that contains data from the sample database:

   ```
   newdemo c:\server1\mirror_demo.db
   ```

3. Run the following command to start the first database server:

   ```
   dbsrv12 -n mirror_server1 -x tcpip(PORT=6871;DOBROAD=no) -su sql
   "c:\server1\mirror_demo.db" -xp on
   ```

   - **-n** Names the database server mirror_server1.
   - **-x** Specifies the port on which the database server runs.
   - **-su** Specifies the password for the utility database.
   - **-xp on** Indicates that the database server is available to participate in a database mirroring system.

4. Connect to the database from Interactive SQL:

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=mirror_server1"
   ```

5. Run the following statements to define the primary and mirror servers:

   ```
   CREATE MIRROR SERVER mirror_demo_primary
   AS PRIMARY
   connection_string='SERVER=mirror_demo_primary;HOST=localhost:
   6871,localhost:6872';

   CREATE MIRROR SERVER mirror_demo_mirror
   AS MIRROR
   connection_string='SERVER=mirror_demo_mirror;HOST=localhost:
   6871,localhost:6872';
   ```

6. Run the following statements to define the mirror_server1 and mirror_server2 as partners in the mirroring system:

   ```
   CREATE MIRROR SERVER mirror_server1
   AS PARTNER
   connection_string='SERVER=mirror_server1;host=localhost:6871'
   state_file='c:\server1\server1.state';

   CREATE MIRROR SERVER mirror_server2
   AS PARTNER
   ```

```
connection_string='SERVER=mirror_server2;host=localhost:6872'
state_file='c:\server2\server2.state';
```

7. Run the following statement to define the arbiter server:

```
CREATE MIRROR SERVER demo_arbiter
AS ARBITER
connection_string ='SERVER=demo_arbiter;HOST=localhost:6870';
```

8. Run the following statement to set the authentication string for the database:

```
SET MIRROR OPTION authentication_string='abc';
```

9. Disconnect from Interactive SQL.

10. Make copies of the database file and transaction log in *c:\server1*, and add them to *c:\server2* by running the following command:

```
dbbackup -c "UID=DBA;PWD=sql;SERVER=mirror_server1;DBN=mirror_demo" c:
\server2
```

11. Run the following command to start the second database server:

```
dbsrv12 -n mirror_server2 -x tcpip(PORT=6872;DOBROAD=no) -su sql
"c:\server2\mirror_demo.db" -xp on
```

    This command line specifies the following dbsrv12 options:

    - **-n** Names the database server mirror_server2.
    - **-x** Specifies the port on which the database server runs.
    - **-su** Specifies the password for the utility database.
    - **-xp on** Indicates that the database server is available to participate in a database mirroring system.

12. Run the following command to start the arbiter database server:

```
dbsrv12 -n demo_arbiter -su sql
-x "TCPIP(PORT=6870;DOBROAD=no)" -xf "c:\arbiter\arbiter.state"
-xa "AUTH=abc;DBN=mirror_demo"
```

    - **-n** Names the database server demo_arbiter.
    - **-su** Specifies the password for the utility database.
    - **-x** Instructs the database server to use TCP/IP communications over port 6870. The other servers also use TCP/IP, but communicate on different ports.
    - **-xf** Specifies the location of the state information file for the arbiter.
    - **-xa** Specifies the names of the database(s) being mirrored and the authentication string (in this case abc) for the arbiter server. This authentication string must be used amongst all the servers (arbiter, primary, and mirror) in a database mirroring system.

13. Start Interactive SQL and connect to the primary server by running the following command:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=mirror_demo_primary;HOST=localhost:
6871,localhost:6872"
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

14. Add data to the database by executing the following statements:

```
CREATE TABLE test (col1 INTEGER, col2 CHAR(32));
INSERT INTO test VALUES(1, 'Hello from server1');
COMMIT;
```

15. Determine which database server you are connected to by executing the following statement:

```
SELECT PROPERTY( 'ServerName' );
```

    The name of the primary server appears.

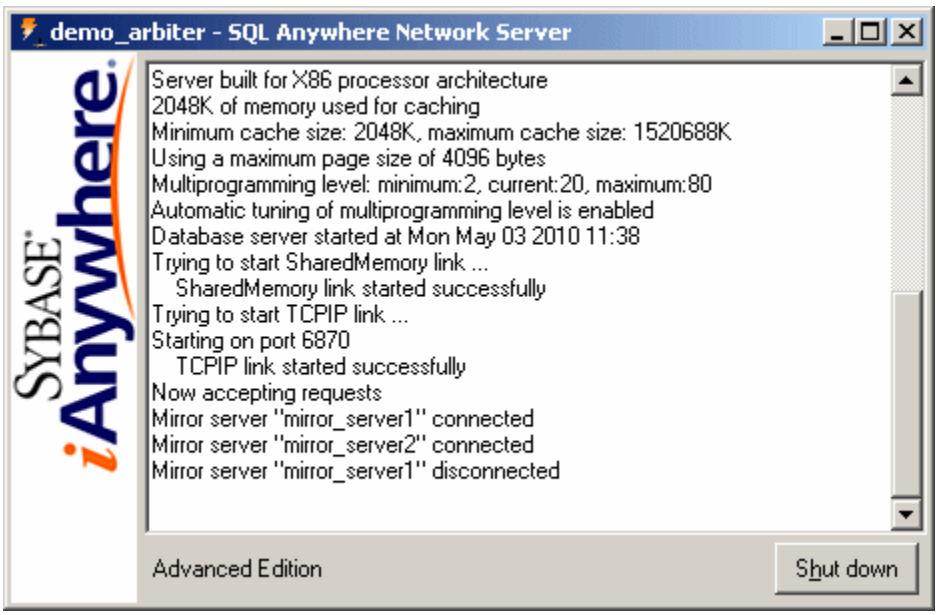
16. Initiate failover. You can do this by stopping the primary server identified in the previous step in one of the following ways:

    - Click **Shut Down** in the database server messages window.
    - Use the Windows Task Manager to end its task.
    - Issue the following command:

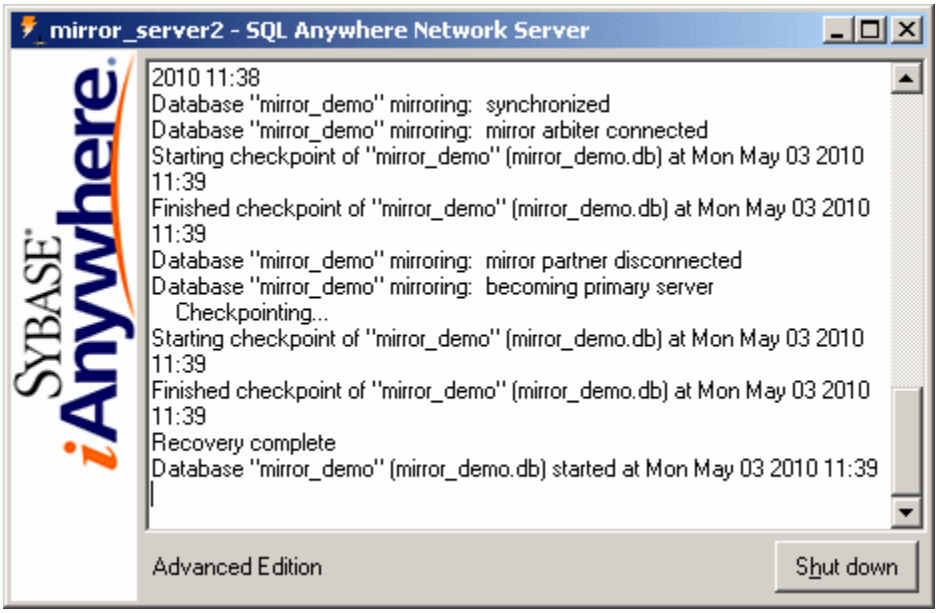
```
dbstop -y -c "UID=DBA;PWD=sql;SERVER=mirror_demo_primary"
```

    If a warning message appears indicating that the database server still has one connection, click Yes to shut it down.

    The arbiter database server messages window displays a message indicating that the primary server is disconnected.



    The database server messages window for mirror_server2 displays a message indicating that it is the new primary server:



17. Close Interactive SQL. If you receive an error message, click **OK**.

18. Restart Interactive SQL by running the following command:

    ```
    dbisql -c "UID=DBA;PWD=sql;SERVER=mirror_demo_primary;HOST=localhost:
    6871,localhost:6872"
    ```

19. Execute the following statement to see that you are now connected to the server that was previously acting as the mirror server:

    ```
    SELECT PROPERTY ( 'ServerName' );
    ```

20. Execute the following statement to verify that all transactions were mirrored to the mirror database:

    ```
    SELECT * FROM test;
    ```

21. Disconnect from Interactive SQL, and then click **Shut Down** on the database server messages window for the arbiter and server2 database servers.

22. (optional) Delete the *c:\server1*, *c:\server2*, and *c:\arbiter* directories.

Tutorial: Using database mirroring with multiple databases sharing an arbiter server

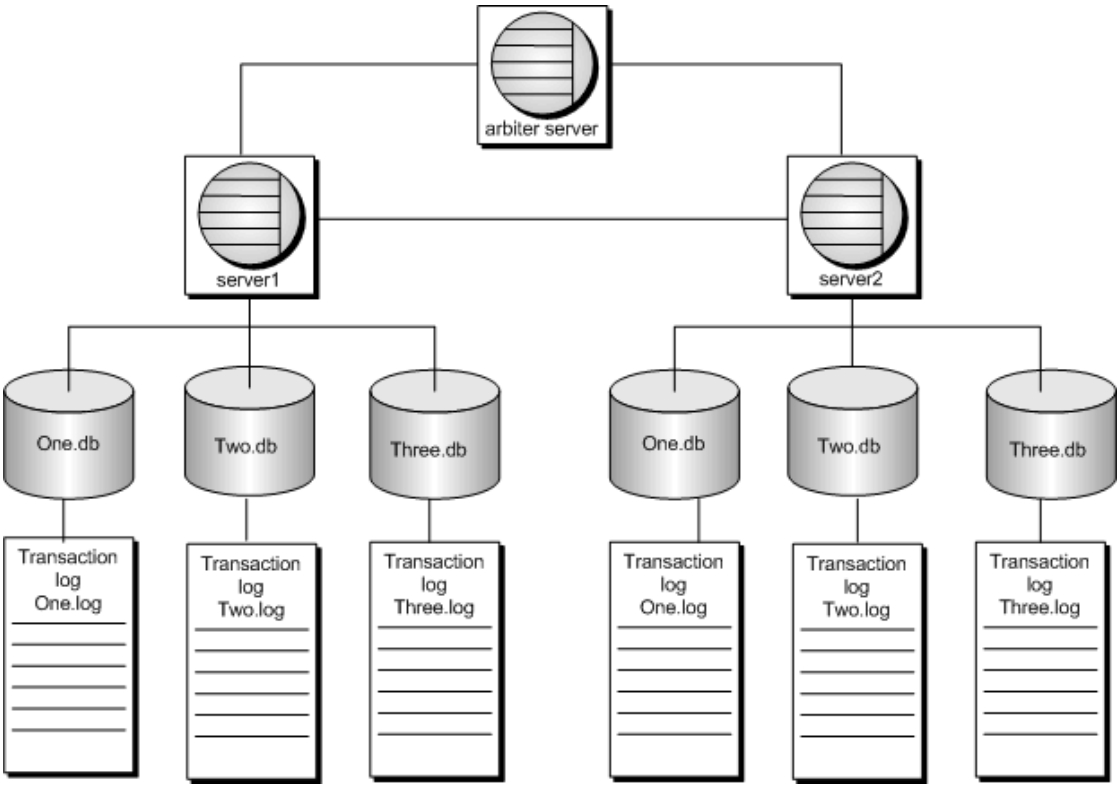
 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Note
If this tutorial is used with database servers running on different computers, references to localhost in the connection strings must be changed to the actual computer names.

There is a sample in *samples-dir\SQLAnywhere\DBMirror* that uses database mirroring with a scale-out system. The sample can be run on one computer, or on multiple computers.

For information about the location of *samples-dir*, see "File locations and installation settings" on page 391.

In this configuration the primary and mirror servers each host three individual databases participating in mirroring systems. All three mirroring systems communicate with the same arbiter server. Each mirroring system uses a unique alternate server name that is specified using the -sn option. With this type of configuration, the primary, mirror, and arbiter servers can all run on separate computers.



If the primary server becomes unavailable, then a role switch occurs and the mirror server takes ownership of the databases. The mirror server becomes the primary server. The client must re-establish a connection to the primary server. The alternate server name is all that needs to be specified to re-establish the connection to the primary server. This configuration also has the ability to protect against failure of a single database. If a database running on the primary server becomes unavailable, then a role switch occurs and the mirror server takes ownership of the failed database. The mirror server becomes the

primary server for only this database. The client must re-establish a connection to the primary server for this database using the alternate server name.

To set up a mirroring system with three databases and one arbiter server

1. Create the following directories:
   - *c:\server1*
   - *c:\server2*
   - *c:\arbiter*
2. Run the following command:

   ```
   newdemo c:\server1\one.db
   ```
3. Run the following command:

   ```
   newdemo c:\server1\two.db
   ```
4. Run the following command from the *c:\server1* directory:

   ```
   newdemo c:\server1\three.db
   ```
5. Start the database server named server1:

   ```
   dbsrv12 -n server1 -x tcpip(PORT=6871) -su sql
   c:\server1\one.db -xp on c:\server1\two.db -xp on c:\server1\three.db -xp
   on
   ```
6. Connect to database one from Interactive SQL and define the required mirroring objects:

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=server1;DBN=one"
   ```
   a. Define server1 as a partner server for database one:

      ```
      CREATE MIRROR SERVER server1
      AS PARTNER
      connection_string='SERVER=server1;host=localhost:6871'
      state_file='c:\server1\server1state.txt';
      ```
   b. Define the database server primary_one as the logical primary server for database one:

      ```
      CREATE MIRROR SERVER primary_one
      AS PRIMARY
      connection_string='SERVER=primary_one;host=localhost:6871,localhost:
      6872';
      ```
   c. Define server2 as a partner server for database one:

      ```
      CREATE MIRROR SERVER server2
      AS PARTNER
      connection_string='SERVER=server2;host=localhost:6872'
      state_file='c:\server2\server2state.txt';
      ```
   d. Define the database server mirror_one as the logical primary server for database one:

      ```
      CREATE MIRROR SERVER mirror_one
      AS MIRROR
      ```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
connection_string='SERVER=mirror_one;host=localhost:6871,localhost:
6872';
```

e. Define the arbiter server:

```
CREATE MIRROR SERVER arbiter
AS ARBITER
connection_string='SERVER=arbiter;HOST=localhost:6870';
```

f. Set the mirroring options for the database mirroring system:

```
SET MIRROR OPTION authentication_string='abc';
```

g. Disconnect from Interactive SQL.

h. Make a backup copy of the database in the *c:\server2* directory:

```
dbbackup -c "UID=DBA;PWD=sql;SERVER=server1;DBN=one" c:\server2
```

7. Connect to database two from Interactive SQL and define the required mirroring objects:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=server1;DBN=two"
```

a. Define server1 as a partner server for database two:

```
CREATE MIRROR SERVER server1
AS PARTNER
connection_string='SERVER=server1;host=localhost:6871'
state_file='c:\server1\server1state.txt';
```

b. Define the database server primary_two as the logical primary server for database two:

```
CREATE MIRROR SERVER primary_two
AS PRIMARY
connection_string='SERVER=primary_two;host=localhost:6871,localhost:
6872';
```

c. Define server2 as a partner server for database two:

```
CREATE MIRROR SERVER server2
AS PARTNER
connection_string='SERVER=server2;host=localhost:6872'
state_file='c:\server2\server2state.txt';
```

d. Define the database server mirror_two as the logical primary server for database two:

```
CREATE MIRROR SERVER mirror_two
AS MIRROR
connection_string='SERVER=mirror_two;host=localhost:6871,localhost:
6872';
```

e. Define the arbiter server:

```
CREATE MIRROR SERVER arbiter
AS ARBITER
connection_string='SERVER=arbiter;HOST=localhost:6870';
```

f. Set the mirroring options for the database mirroring system:

```
SET MIRROR OPTION authentication_string='def';
```

g. Disconnect from Interactive SQL.

h. Make a backup copy of the database in the *c:\server2* directory:

```
dbbackup -c "UID=DBA;PWD=sql;SERVER=server1;DBN=two" c:\server2
```

8. Connect to database three from Interactive SQL and define the required mirroring objects:

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=server1;DBN=three"
   ```

   a. Define server1 as a partner server for database three:

      ```
      CREATE MIRROR SERVER server1
      AS PARTNER
      connection_string='SERVER=server1;host=localhost:6871'
      state_file='c:\server1\server1state.txt';
      ```

   b. Define the database server primary_three as the logical primary server for database three:

      ```
      CREATE MIRROR SERVER primary_three
      AS PRIMARY
      connection_string='SERVER=primary_three;host=localhost:6871,localhost:
      6872';
      ```

   c. Define server2 as a partner server for database three:

      ```
      CREATE MIRROR SERVER server2
      AS PARTNER
      connection_string='SERVER=server2;host=localhost:6872'
      state_file='c:\server2\server2state.txt';
      ```

   d. Define the database server mirror_three as the logical primary server for database three:

      ```
      CREATE MIRROR SERVER mirror_three
      AS MIRROR
      connection_string='SERVER=mirror_three;host=localhost:6871,localhost:
      6872';
      ```

   e. Define the arbiter server:

      ```
      CREATE MIRROR SERVER arbiter
      AS ARBITER
      connection_string='SERVER=arbiter;HOST=localhost:6870';
      ```

   f. Set the mirroring options for the database mirroring system:

      ```
      SET MIRROR OPTION authentication_string='ghi';
      ```

   g. Disconnect from Interactive SQL.

   h. Make a backup copy of the database in the *c:\server2* directory:

      ```
      dbbackup -c "UID=DBA;PWD=sql;SERVER=server1;DBN=three" c:\server2
      ```

9. Start the database server named server2:

   ```
   dbsrv12 -n server2 -x tcpip(PORT=6872)
   -su sql c:\server2\one.db -xp on c:\server2\two.db -xp on c:
   \server2\three.db -xp on
   ```

10. Start the arbiter server.

    ```
    dbsrv12
    -n arbiter
    -su sql
    -x tcpip(port=6870)
    -xf c:\arbiter\arbiterstate.txt
    -xa "AUTH=abc,def,ghi;DBN=one,two,three"
    ```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

After starting server2, the server1 database server messages window shows that server1 is the primary server in the mirroring system for databases one, two, and three. The messages also indicate that the mirror databases for one, two, and three (partners) are connected to server1.

The arbiter messages show that both server1 and server2 are connected.

11. Run the following command to start Interactive SQL and connect to database one on the primary server:

```
dbisql -c "UID=DBA;PWD=sql;Server=primary_one;LINKS=TCPIP"
```

12. Add sample data to the SQL Anywhere sample database by executing the following statements:

```
CREATE TABLE test (col1 INTEGER, col2 CHAR(32));
INSERT INTO test VALUES(1, 'Hello from server1');
COMMIT;
```

13. Determine which database server you are connected to by executing the following statement:

```
SELECT PROPERTY( 'ServerName' );
```

The name of the primary server appears.

14. Disconnect from Interactive SQL.

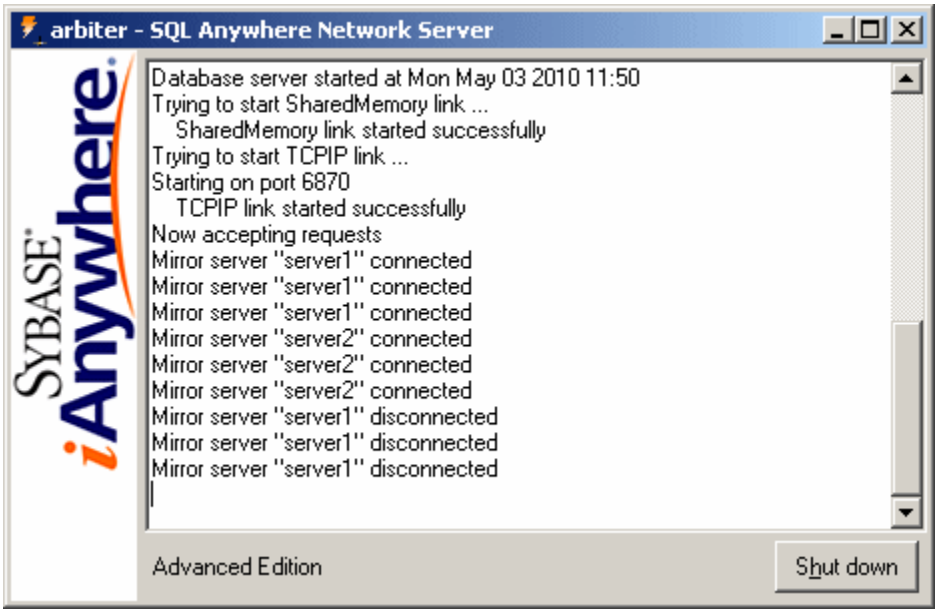
15. Initiate failover. You can do this by stopping the primary server in one of the following ways:

- Click **Shut Down** in the database server messages window.
- Use the Windows Task Manager to end its task.
- Issue the following command:

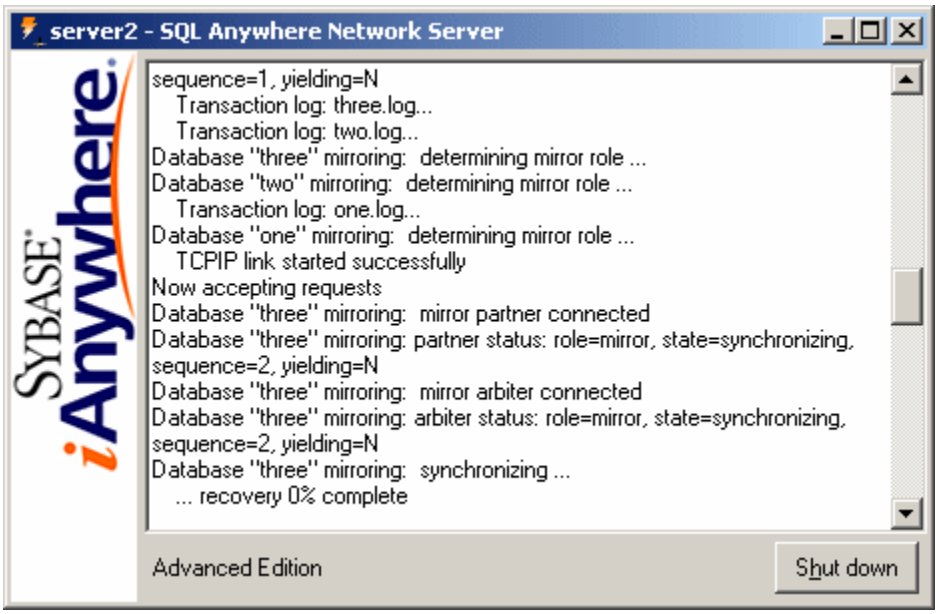
```
dbstop -y -c "UID=DBA;PWD=sql;Server=server1"
```

If a warning message appears indicating that the database server still has one connection, click Yes to shut it down.

The arbiter database server messages window displays a message indicating that the primary server is disconnected.



The database server messages window for server2 displays a message indicating that it is the new primary server:



16. Restart Interactive SQL by running the following command:

```
dbisql -c "UID=DBA;PWD=sql;Server=primary_one;LINKS=tcpip"
```

17. Execute the following statement to see that you are now connected to server2:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
SELECT PROPERTY ( 'ServerName' );
```

18. Execute the following statement to verify that all transactions were mirrored to the mirror database:

    ```
    SELECT * FROM test;
    ```

19. Disconnect from Interactive SQL, and then click **Shut Down** on the database server messages window for the arbiter and server2 database servers.

20. (optional) Delete the *c:\server1*, *c:\server2*, and *c:\arbiter* directories.

Setting up a database mirroring system

When starting database servers that will be participating in a mirroring system, you must specify **-xp on**. It is recommended that you also include the -su option to specify the password for the utility database. Then, you can use the utility database to shut down the database server, or force the mirror server to become the primary server if necessary. See "Stopping a database server in a mirroring system" on page 969 and "Forcing a database server to become the primary server" on page 968.

For information about upgrading SQL Anywhere or rebuilding a database involved in a database mirroring system, see "Upgrading SQL Anywhere software and databases in a database mirroring system" [*SQL Anywhere 12 - Changes and Upgrading*].

Note
When setting up a database mirroring system, in all the examples below, localhost and the port number need to be changed to the computer name and port where the corresponding database server will be running.

To set up a mirroring system

1. Start the database that is going to be mirrored on a database server with the **-su** and **-xp on** options. For example:

   ```
   dbsrv12 -n mirror_server1 -x tcpip(PORT=6871;DOBROAD=no) -su sql
   "c:\server1\mymirrordb.db" -xp on
   ```

   See "-su dbeng12/dbsrv12 server option" on page 225 and "-xp dbsrv12 database option" on page 263.

2. Define the partner servers and arbiter server for the database by using the CREATE MIRROR SERVER statement. You must define one database server as the primary server and one database server as the mirror server, as well as an arbiter server. You must also define the primary and mirror servers as partners in the database mirroring system.

   The first statement defines the name that clients use to connect to the database server that is acting as the primary server in the database mirroring system. The second statement defines mirror_server1 as a partner server in the database mirroring system.

   ```
   CREATE MIRROR SERVER myprimary
   AS PRIMARY
   connection_string='SERVER=myprimary;HOST=localhost:6871,localhost:6872';
   ```

```
CREATE MIRROR SERVER mirror_server1
AS PARTNER
connection_string='SERVER=mirror_server1;host=localhost:6871'
state_file='c:\server1\server1.state';
```

The following SQL statements define the name of the database server that is acting as the mirror server in the database server, and also define the second partner server in the database mirroring system:

```
CREATE MIRROR SERVER mymirror
AS MIRROR
connection_string='SERVER=mymirror;HOST=localhost:6871,localhost:6872';

CREATE MIRROR SERVER mirror_server2
AS PARTNER
connection_string='SERVER=mirror_server2;host=localhost:6872'
state_file='c:\server2\server2.state';
```

Note
The roles of primary and mirror are necessary for configuring the database servers in the system: the names that you give these servers are used as alternate server names when clients connect to the database servers. Either partner server can act as the primary or mirror server.

The following SQL statement defines the arbiter server for the database mirroring system:

```
CREATE MIRROR SERVER myarbiter
AS ARBITER
connection_string ='SERVER=myarbiter;HOST=localhost:6870';
```

See "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*].

3. Set mirroring options for the mirroring system. You must specify an authentication string. For example:

   ```
   SET MIRROR OPTION authentication_string='abc';
   ```

   See "SET MIRROR OPTION statement" [*SQL Anywhere Server - SQL Reference*].

4. Make a copy of the database and current transaction log on the computer where the second database server will be running.

   If you stop the database server running the database you want to mirror, you can copy files; otherwise, use the BACKUP DATABASE statement or the Backup utility (dbbackup). See "BACKUP statement" [*SQL Anywhere Server - SQL Reference*] and "Backup utility (dbbackup)" on page 767.

5. Start the second database server in the database mirroring system:

   ```
   dbsrv12 -n mirror_server2 -x tcpip(PORT=6872;DOBROAD=no) -su sql
   "c:\server2\mymirrordb.db" -xp on
   ```

6. Start the arbiter server.

   ```
   dbsrv12 -n myarbiter -su sql
   -x "TCPIP(PORT=6870;DOBROAD=no)" -xf "c:\arbiter\arbiter.state"
   -xa "AUTH=abc;DBN=mymirrordb"
   ```

   Clients can now connect to the mirrored database.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Tip
You can check the status of the database servers in a database mirroring system by connecting to the primary database from Sybase Central. Database mirroring information is available on the **Health and Statistics** pane. See "Monitoring database health and statistics" on page 695.

Configuring the connection strings between mirror servers

When you create the mirror servers, you can configure the connection strings between the database servers in the mirroring system so that broadcasts are avoided by specifying DoBroadcast=none. This allows the database server to determine more quickly that another database server is unavailable. If the connection string also specifies the IP address or computer name, as well as the port, for the database server you want to connect to, a TCP/IP connection can be made without the need for a broadcast. If the server is not available, the connection fails, and there is no need for a timeout. Using the Host connection parameter also avoids the need to specify a timeout for the connection. For example:

```
CREATE MIRROR SERVER mirror_one
AS PRIMARY
connection_string='SERVER=mirror_one;host=localhost:6871,localhost:6872';
```

For more information, see:

- "Host connection parameter" on page 291
- "DoBroadcast (DOBROAD) protocol option" on page 320

Connecting to a mirrored database server

When connecting to a mirrored database, clients must use the server name that was specified for the primary database server in the database mirroring system. For example, if the primary database server is named myprimary, clients specify the connection parameter Server=myprimary in their connection string:

```
...UID=user12;PWD=x92H4pY;Server=myprimary;HOST=localhost:6871,localhost:
6872...
```

If the primary and mirror servers are running on different subnets, then you must specify a list of IP addresses that the client should use to connect to the primary server. For example:

```
...UID=user12;PWD=x92H4pY;Server=myprimary;LINKS=tcpip(HOST=ip1,ip2...)...
```

You may also want to specify the RetryConnectionTimeout connection parameter to control how long clients keep retrying the connection attempt to the primary server. See "RetryConnectionTimeout (RetryConnTO) connection parameter" on page 306.

If you are having trouble locating the server to which clients need to connect, try the following:

1. Specify the host name of the computers running the primary and mirror servers. For example, if they are running on computers named MirrorServ1 and MirrorServ2, you can use `HOST=MirrorServ1,MirrorServ2` in the client connection string.

2. Register the servers with LDAP. See "Connecting using an LDAP server" on page 81.

3. Use the SQL Anywhere Broadcast Repeater utility (dbns12) to locate the servers. This utility listens for broadcasts and responses on one subnet, and then re-broadcasts them on another subnet. See "Broadcast Repeater utility (dbns12)" on page 772.

Determining the initial primary server

When you first set up a database mirroring system and there are no state information files, and the copies of the database and transaction log are identical, both servers are eligible to act as the primary. In this situation, the server names are compared, and the server with the lower name acts as primary. For example, the name server1 is lower than server2.

For the initial startup, both servers must be running and connected for them to agree on roles; the presence of an arbiter is not enough since the prior state information recorded in the state information files does not exist.

During a normal startup, the following inputs affect which server becomes the primary server:

- the contents of the state information files
- the transaction log position on each database server
- the designation of a preferred primary server

If a database with no mirroring definitions is started on a database server (for example, S1) started with **-xp on**, and the mirroring definitions are then made, S1 would be the initial primary server for the database.

See also

- "State information files" on page 952

Specifying a preferred database server

In a database mirroring system, you can identify one of the two partner servers as the preferred server. If all the database servers are running, then the preferred server becomes the primary server and takes ownership of the database. If the server that is marked as preferred becomes unavailable, then the server that was acting as the mirror server becomes the primary server. When the preferred server restarts, it obtains any transaction log entries it does not already have from the current primary server. It then asks the current primary server to relinquish ownership of the database. The servers then change roles, with the preferred server becoming the primary server and the other server becoming the mirror server. Any connections to the database on the non-preferred server are lost when the database ownership changes.

You specify a preferred server by adding PREFERRED='YES' to the CREATE MIRROR SERVER statement that defines the partner server. For example:

```
CREATE MIRROR SERVER mirror_server1
AS PARTNER
connection_string='SERVER=mirror_server1;host=localhost:6871'
state_file='c:\server1\server1.state'
preferred='YES';
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]
- "ALTER MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]
- "Initiating failover on the primary server" on page 968
- "Choosing a database mirroring mode" on page 950

Configuring read-only access to a database running on the mirror server

When using database mirroring, you can access the database running on the mirror server using a read-only connection. This functionality is useful if you want to offload reporting or other operations that require read-only access to this database.

In a mirroring system, you do not necessarily know which database server is acting as the primary server and which one is acting as the mirror server. If you want to be able to connect to the database running on the mirror server, define a mirror server with type MIRROR. This allows connections to find the mirror server by providing a server name that is used to access the read-only mirror database. A server name specified in this way is only active when the database server is acting as mirror for the database. For example, the following statement defines the name mysamplemirror as an alternate server name for use when connecting to the database running on the mirror server:

```
CREATE MIRROR SERVER mysamplemirror
AS MIRROR
connection_string='SERVER=mysamplemirror;HOST=winxp-1:6871,winxp-2:6872';
```

Any attempt to make a change to the database results in an error, which is the same behavior as when a database is started as read-only using the -r option. You can perform operations on temporary tables, but events are not fired on the mirror database. Event firing only starts after failover from the primary server to the mirror server takes place. The DatabaseStart and MirrorFailover events fire at that time, if they are defined. For more information, see "Understanding system events" on page 935.

Connections to the mirror database are maintained if failover occurs and the mirror server becomes the primary server. After failover, a connection can make changes to the database. You can query the value of the ReadOnly database property to determine whether the database you are connected to is updatable:

```
SELECT DB_PROPERTY( 'ReadOnly' );
```

See also

- "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]
- "ALTER MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]
- "ReadOnly database property" on page 673

Running queries against the mirror database

Queries that are executed against the mirror database can place locks, depending on the isolation level specified. If locks interfere with operations being applied from the primary server, then the connections

holding the locks have their transactions rolled back and any open cursors for those connections are closed. Applications running at isolation level 0 do not add row locks, but still acquire schema locks. If the schema locks interfere with operations being applied from the primary server, the transaction on the mirror database is rolled back.

Applications that require a consistent view of the database (and so cannot use isolation level 0) should consider using snapshot isolation. To do so, the allow_snapshot_isolation option must be set to On. This option takes effect on both the primary server and the mirror server, so the costs associated with snapshot isolation need to be considered.

Connections to the mirror database are affected by transactions against the primary server, since those operations are then processed and applied by the mirror server. There can be a small delay between the time an update on the primary server is committed and the time that the update is available on the mirror server. Normally this delay is short, but you should keep this in mind when you are accessing the database running on the mirror server.

See also

- "Snapshot isolation" [*SQL Anywhere Server - SQL Usage*]
- "allow_snapshot_isolation option" on page 506

Forcing a database server to become the primary server

In situations where you need to force the primary server to shut down (for example, if you are replacing the computer it is running on), you can force the mirror server to become the primary server when it would not otherwise take ownership of the database by using the ALTER DATABASE statement.

You must connect to the utility database on the mirror database server to use this feature. To connect to the utility database, you must specify the -su option in the command to start the mirroring servers. The following command forces the mirror server for the database *mymirroreddb.db* to become the primary server:

```
ALTER DATABASE mymirroreddb FORCE START;
```

The FORCE START clause forces a database server that is currently acting as the mirror server to take ownership of the database. This statement must be executed while connected to the utility database on the mirror server. See "Connecting to the utility database" on page 29.

If you want to force a failover from the primary server to the mirror server, you can:

- stop the primary server
- execute ALTER DATABASE SET PARTNER FAILOVER while connected to the database on the primary server (this statement causes the primary server to restart the database and become the mirror)

See also

- "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*]

Initiating failover on the primary server

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can initiate a database mirroring failover from the primary server to the mirror server by executing the following statement:

```
ALTER DATABASE SET PARTNER FAILOVER;
```

This statement is an alternative to specifying a preferred server, and can be used with logic that controls when ownership of the database is given to a specific database server. For example, you may want to initiate failover based on the availability of the partner server (determined by the value of the PartnerState database property), or the number of connections to the database (determined by the value of the ConnCount database property).

When this statement is executed, any existing connections to the database are closed, including the connection that executed the statement. If the statement is contained in a procedure or event, other statements that follow it may not be executed. The permissions required to execute this statement are controlled by the -gk server option.

See also

- "Specifying a preferred database server" on page 966
- "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "-gk dbeng12/dbsrv12 server option" on page 187
- "ConnCount database property" on page 663
- "PartnerState database property" on page 672

Stopping a database server in a mirroring system

There may be situations where you need to stop the primary, mirror, or arbiter server. You can use the Stop Database utility (dbstop) to do this.

You must use a connection to the utility database to stop the server, so it is recommended that you include the -su server option when starting the database server. See "Using the utility database" on page 28.

To stop a primary, mirror, or arbiter server

- Issue a dbstop command to stop the database server.

  For example, the following command stops a database server named mirror_server_1:

  ```
  dbstop -c "UID=DBA;PWD=sql;DBN=utility_db;LINKS=tcpip" mirror_server_1
  ```

Dropping mirror servers

You can use a DROP MIRROR SERVER statement to drop mirror servers. It is recommended that you do not drop a mirror server that is running as part of a database mirroring system.

When you set up a database mirroring system, mirror servers that are eligible to become the primary server have two definitions (CREATE MIRROR SERVER ... AS PARTNER and CREATE MIRROR

SERVER ... AS PRIMARY or CREATE MIRROR SERVER ... AS MIRROR). You may need to drop both the role and partner server definitions from the database, depending on your reason for dropping the mirror server.

Note
If you want to keep the same mirror server name but change its settings, you can use the CREATE OR REPLACE MIRROR SERVER statement or the ALTER MIRROR SERVER statement. See "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*] and "ALTER MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*].

To drop a mirror server (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a DROP MIRROR SERVER statement to drop the mirror server. For example:

```
DROP MIRROR SERVER mirror-server-name;
```

See also

- "DROP MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]

Recovering from primary server failure

The steps for recovering from primary server failure depend on the synchronization mode you are using for your database mirroring system.

If you are running in synchronous mode, then all the transactions that are present on the primary server are also guaranteed to be committed on the mirror server. The mirror server can take over as the new primary server without any user intervention.

In asynchronous or asyncfullpage mode, failover from the primary server to the mirror server is not automatic because the mirror server may not have all committed transactions that were applied on the primary server. Unless you specified that autofailover should take place, when using one of the asynchronous modes, a mirror server, by default, cannot take ownership of a database when the primary fails. When the failed server is restarted, it detects whether transactions were lost. If transactions were lost, it writes a message to the database server message log and shuts down the database.

When starting the original mirror server as the new primary server, you have two options for getting the database files on both servers into the same state:

- Copy the database and transaction log files from the original primary server to the mirror server and then start the mirror server as the new primary server. You can force a server to be the primary server using the ALTER DATABASE statement. See "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*].
- Perform a backup (using dbbackup) on the original mirror server. Copy the files to the original primary server, and then start the database servers.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Database mirroring and transaction log files

When a partner server starts, it examines all the transaction log files in the same directory as the current transaction log file and determines which ones need to be applied. The database server then applies the operations in these transaction logs to the database before determining whether to act as the primary or mirror server.

Once a server takes on the role of mirror, it starts receiving transaction log pages from the primary server. When a transaction log rename occurs on the primary, the rename is also performed on the mirror. The mirror then writes new transaction log pages to a new file with the name specified for the transaction log.

Transaction log files can be deleted periodically on the primary. Each time a transaction log file is renamed, the mirror is notified about which transaction log file is the oldest surviving file on the primary. Any transaction log files older than this are deleted on the mirror.

Because a mirror server may not be available when a backup is performed against the primary server that requests a transaction log truncation, deletion of transaction logs on the primary must be performed using a different method than truncating the transaction log (such as a scheduled event that uses xp_cmdshell to delete files more than one week old).

Database mirroring system events

The following system events are supported for database mirroring:

- **MirrorFailover** This event fires each time a database server takes ownership of the mirrored database. For example, it fires when a server first starts and determines that it should own the database. It also fires when a server previously acting as the mirror determines that the primary server has gone down and, after consulting with the arbiter, determines that it should take ownership.
- **MirrorServerDisconnect** When the connection between the primary server and mirror server or arbiter server is lost, the MirrorServerDisconnect event fires. Within the handler for this event, the value of EVENT_PARAMETER('MirrorServerName') is the name of the server whose connection was lost.

Events are not fired on a server that is currently acting as the mirror server. As well, mirroring events cannot be defined to execute on an arbiter, since events only run in the context of the database in which they are defined, and the arbiter does not use a copy of the database being mirrored.

You can use these events as a mechanism to send notification via email that action may be required on the mirror database. These events may not fire in all situations that result in the database running on the primary server becoming unavailable. For example, a power outage affecting both the primary and mirror servers would prevent either of these events from being fired. If this type of monitoring is required, it can be implemented on a separate computer via a scripting language by calling dbping to periodically connect to the mirror database. See “Ping utility (dbping)” on page 826.

The following example creates an event that notifies an administrator when failover occurs:

```
CREATE EVENT mirror_server_unavailable
TYPE  MirrorServerDisconnect
```

```
HANDLER
BEGIN
 CALL xp_startmail ( mail_user ='George Smith',
                     mail_password ='mypwd' );
 CALL xp_sendmail( recipient='DBAdmin',
    subject='Mirror server disconnect occurred',
    "message"='The following server is unavailable in the mirroring system: '
    || event_parameter( 'MirrorServerName' ) );
 CALL xp_stopmail ( );
END;
```

See also

- "Understanding system events" on page 935

Database mirroring and performance

Ideally, the computers running the primary and mirror servers should be configured with similar hardware (processor, disk, memory, and so on). At any given time, the database server running on either computer can be acting as the primary server for the database being mirrored. The mirror server utilization will typically be low, depending on update activity on the primary.

Query performance against the primary server is not affected by mirroring. The performance of transactions that update the database depends on the size of the transaction and the frequency of commits. A mirror server operating in asynchronous mode has better performance than one in synchronous mode, but is still slower than a database server that is not participating in a mirroring system. Performance is highly dependent on the speed of the network connection between the partner servers.

Database mirroring and backups

Although database mirroring can help minimize the risk of data loss, it is still recommended that you back up and validate databases that are participating in a database mirroring system.

You can use the BACKUP DATABASE statement to perform a backup relative to the database server. The BACKUP DATABASE statement is executed on the primary database server, so the file name that is provided should specify a network drive or UNC name that is consistent for both the primary and mirror database servers. See "BACKUP statement" [*SQL Anywhere Server - SQL Reference*].

Alternatively, you can perform client-side backups using the dbbackup utility. See "Backup utility (dbbackup)" on page 767.

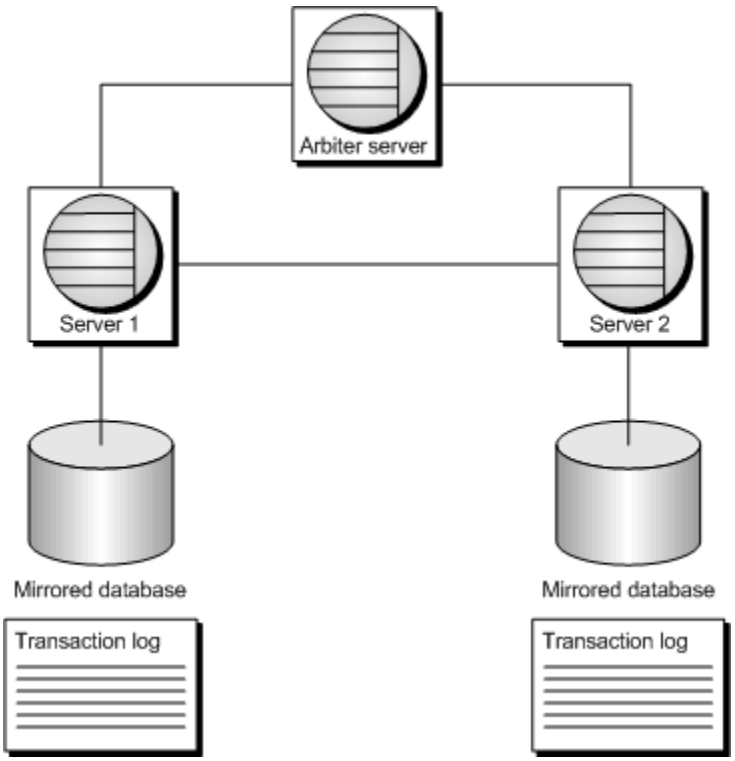
See also

- "Backup and data recovery" on page 887
- "Validating databases" on page 927

Database mirroring scenarios

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The following scenarios help you understand what happens when a server becomes unavailable in a mirroring system. The scenarios use the following database mirroring configuration, which consists of Server 1, Server 2, and an arbiter server running in synchronous mode:



At any time, you can use the MirrorState, PartnerState, and ArbiterState database properties to determine the status of the database servers in the mirroring system. See "Database properties" on page 659.

Scenario 1: Primary server becomes unavailable

1. The primary server (Server 1) becomes unavailable. All clients are disconnected.
2. The arbiter and Server 2 detect that Server1 is no longer available.
3. The arbiter and Server 2 reach quorum, and Server 2 becomes the primary server.
4. Server 2 begins accepting client connections.

In this scenario, if you are running in asynchronous or asyncfullpage mode and did not specify that autofailover should occur, then you may need to make a copy of the database and restart the server that is still operational before clients can connect again.

For more information about recovering when the primary server becomes unavailable, see "Recovering from primary server failure" on page 970.

Scenario 2: Primary server becomes unavailable and then restarts

1. The primary server (Server 1) becomes unavailable. All clients are disconnected.
2. The arbiter and mirror server (Server 2) detect that the primary server (Server 1) is no longer available.
3. The arbiter and Server 2 reach quorum, and Server 2 becomes the primary server.
4. Server 2 begins accepting client connections.
5. Server 1 comes back online and reconnects to Server 2 and the arbiter.
6. Server 1 requests quorum, but Server 2 is already the primary server.
7. Server 1 is the mirror server, and waits for changes from Server 2.
8. Server 2 sends changes to Server 1.

Should Server 2 become unavailable before Server 1 has received all the transactions from Server 2, Server 1 will not be able to reach a synchronized state. It must wait for Server 2 to become available again so it can obtain and apply the transactions it does not yet have.

For more information about recovering when the primary server becomes unavailable, see "Recovering from primary server failure" on page 970.

Scenario 3: Mirror server becomes unavailable

1. The mirror server (Server 2) becomes unavailable.
2. The arbiter and Server 1 detect that the mirror server (Server 2) is no longer available.

   Client connections are not affected. They can continue to connect to the primary server. However, if either Server 1 or the arbiter server becomes unavailable, the clients will not be able to connect.

Scenario 4: Mirror server becomes unavailable and then restarts

1. The mirror server (Server 2) becomes unavailable.
2. Client connections are not affected because there is no change in availability. They can continue to connect to the primary server. However, if either Server 1 or the arbiter server becomes unavailable, then clients will not be able to connect.
3. Server 2 comes back online and reconnects to Server 1 and the arbiter.
4. Server 2 requests quorum, but Server 1 is already the primary server.
5. Server 2 is the mirror server, and waits for changes from Server 1.
6. Server 1 sends changes to Server 2.

   Client connections are not affected because there is no change in availability. They continue connecting to Server 1.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Scenario 5: Arbiter becomes unavailable

1. Server 1 (primary server) and Server 2 (mirror server) detect that the arbiter is gone.

2. Both servers remain available. Clients are not disconnected.

   When the arbiter comes back online, Server 1 and Server 2 will detect it, and begin communicating with it. There is no change in database availability for clients.

   If Server 1 or Server 2 becomes unavailable when there isn't an arbiter server, the other server cannot reach quorum by itself, and the database will not be available.

Scenario 6: Arbiter restarts

1. Arbiter comes back online and reconnects to Server 1 and Server 2.

   Client connections are not affected because there is no change in availability.

Using the SQL Anywhere Veritas Cluster Server agents

Separately licensed component required
The SQL Anywhere Veritas Cluster Server agents require a separate license. See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

A **cluster** is a group of computers, called **nodes**, that work together to run a set of applications. Clients connecting to applications running on a cluster treat the cluster as a single system. If a node fails, other nodes in the cluster can automatically take over the services provided by the failed node. Clients may see a slight disruption in availability (the time it takes to resume the services on the remaining nodes), but are otherwise unaware that the node has failed.

When you use clustering with SQL Anywhere, any uncommitted transactions are lost when a database or database server fails over to another node in the cluster, and clients must reconnect to the database after failover occurs.

SQL Anywhere supports a variety of cluster environments where the cluster software can make any application into a generic resource subject to automatic failover so that high availability can be provided. However, only the database server process can be failed over, and the monitoring and control processes are limited.

For more information, see http://www.sybase.com/detail?id=1034743.

Most cluster software provides an API for creating custom resources tailored to a specific application. SQL Anywhere includes two custom failover resources for Veritas Cluster Server: SAServer and SADatabase. The SAServer agent is responsible for database server failover, while the SADatabase agent is responsible for the failover of a specific database file. You can use one or both agents, depending on your application.

Your systems must be set up as follows to use the SQL Anywhere Veritas Cluster Server agents:

- You must use Veritas Cluster Server 4.1 or later.
- SQL Anywhere must be installed identically on each system node within the cluster.
- Database files must be stored on a shared storage device that is accessible to all systems within the cluster.
- The utility database password must be the same for all systems within the cluster.

  The SADatabase agent uses the utility database to start and stop specific database files. All systems participating in the cluster must have the same utility database password. You can set the utility database password by specifying the -su server option when starting the database server.

  On Unix, the VCS agent is installed in *install-dir/vcsagent/saserver*.

There are three ways to configure and add a new agent to Veritas Cluster Server:

1. Using the Cluster Manager.
2. Using command line utilities.
3. Using a text editor and editing the *main.cf* configuration files.

The instructions in the following sections use the Cluster Manager.

For information about the available utilities, see *Veritas Cluster Server Administration Guide*.

If you want to configure *main.cf* manually using a text editor, you must stop all Veritas Cluster Server services before editing the *main.cf* file. Otherwise, the changes do not take effect.

Configuring the SAServer agent

The SAServer agent controls the failover of a SQL Anywhere database server to another node in the cluster.

To set up the SAServer agent

1. Shut down all SQL Anywhere database servers running on nodes in the cluster.
2. Choose a node in the cluster and create a directory named *SAServer* under the *%VCS_HOME%\bin* directory on that node. You will see other Veritas Cluster Server agents within this folder (such as NIC and IP).
3. Copy the following files from the *install-dir\VCSAgent\SAServer* directory to the *SAServer* directory you created in Step 2:
   - *Online.pl*
   - *Offline.pl*
   - *Monitor.pl*
   - *Clean.pl*
   - *SAServer.xml*

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

4. Copy the file *%VCS_HOME%\bin\VCSdefault.dll* into the *%VCS_HOME%\bin\SAServer* directory and rename it to *SAServer.dll*.
5. Copy the file *install-dir\VCSAgent\SAServer\SAServerTypes.cf* into the *%VCS_HOME%\conf\config* directory.
6. Repeat Steps 1-5 for all other nodes in the cluster.
7. Start the Veritas Cluster Server Manager and enter your user name and password to connect to the cluster.
8. Add the SAServer agent:
   a. Choose **File** » **Import Types**.
   b. Navigate to *%VCS_HOME%\conf\config\SAServerTypes.cf*, and then click **Import**.

To set up a database server for failover using the SAServer agent

1. Start the Veritas Cluster Server Manager and enter your user name and password to connect.
2. Add SAServer as a resource to a service group:
   a. Choose **Edit** » **Add** » **Resource**.
   b. In the **Resource Type** list, choose **SAServer**.

      On Windows, if SAServer does not appear in the **Resource Type** list under Windows, you may have to add the *SAServer.xml* file to the *%VCS_ROOT%\cluster manager\attrpool\Win2K\400* and restart the cluster services.
   c. In the **Resource Name** field, type a name.
   d. Add the following attribute values to the following attributes:
      - **cmdStart** dbsrv12 -x tcpip *database-file-on-shared-disk* -n *server-name*
      - **cmdMonitor** dbping -c "Server=*server-name*"
      - **cmdStop** dbstop -c *user-id,password* -y
   e. Select **Enabled**.

      This indicates that the resource is ready to be used.
   f. Click **OK**.
3. Ensure that the resource dependencies are configured correctly. There are other resources that must be started and grouped together before SAServer can be started, such as the shared disk resources and the IP address resources.
4. Right-click the service group and choose **Online** » *node-name*, where *node-name* is the name of the computer in the cluster on which you want the resource to run.

   The service group is now online.

Testing the SAServer agent

The following steps describe how you can test a failover situation for the SAServer agent.

To test SAServer agent failover

1. Connect to the database from Interactive SQL. For example:

   ```
   dbisql -c "UID=DBA;PWD=sql;Server=VCS;LINKS=tcpip"
   ```

2. Execute the following query:

   ```
   SELECT * FROM Departments;
   ```

   The query should execute without errors.

3. Shut down the system running the database server.

   Failover should occur, and all resources should start on the alternate server.

4. Reconnect from Interactive SQL using the same connection string and executing the query again. You should be able to connect and execute the query successfully.

Configuring the SADatabase agent

The SADatabase agent controls the failover of a SQL Anywhere database to another node in the cluster.

To set up the SADatabase agent

1. Shut down all SQL Anywhere database servers running on nodes in the cluster.

2. Create a directory named *%VCS_HOME%\bin\SADatabase* on one of the nodes in the cluster.

3. Copy the following files from the *install-dir\SADatabase* directory to the *%VCS_HOME%\bin\SADatabase* directory you created in Step 2:

   - *Online.pl*
   - *Offline.pl*
   - *Monitor.pl*
   - *Clean.pl*
   - *SADatabase.xml*

4. Copy the file *%VCS_HOME%\bin\VCSdefault.dll* into the *%VCS_HOME%\bin\SADatabase* directory and rename it to *SADatabase.dll*.

5. Copy the file *install-dir\SADatabase\SADatabaseTypes.cf* into the *%VCS_HOME%\conf\config* directory.

6. Repeat Steps 1-5 for all systems participating in the cluster.

7. Start the Veritas Cluster Server Manager and enter your user name and password to connect to the cluster.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

8. Add the SADatabase agent:
   a. From the **File** menu, choose **Import Types**.
   b. Navigate to *%VCS_HOME%\conf\config*, and click **Import**.

To set up a database for failover using the SADatabase agent

1. Add SADatabase as a resource to the service group:
   a. From the **Edit** menu, choose **Add** » **Resource**.
   b. From the **Resource Type** list, choose **SADatabase**.

      On Windows, if SADatabase does not appear in the **Resource Type** list, you may have to add the *SADatabase.xml* file to the *%VCS_ROOT%\cluster manager\attrpool\Win2K\400* and restart the cluster services.
   c. In the **Resource Name** field, type a name.
   d. Add the specified values to the following attributes by clicking the button in the **Edit** column for each attribute:
      - **DatabaseFile** The location of the database file, for example, *E:\demo.db*.
      - **DatabaseName** A name for the database.
      - **ServerName** A name for the database server. A different server name can be supplied on each system within the cluster. The scope of the attribute should be Per System, not Global.
      - **UtilDBpwd** The utility database password used for all systems within the cluster.
   e. Select **Enabled**.

      This indicates that the resource is ready to be used.
   f. Click **OK**.

2. Ensure that the resource dependencies are configured correctly. There are other resources that must be started/grouped together before SADatabase can be started, such as the shared disk resources and the IP address resources.

3. Right-click the service group, and choose **Online** » *node-name*, where *node-name* is the name of the computer in the cluster on which you want the resource to run.

   The service group is now online.

Testing the SADatabase agent

The following steps describe how you can test a failover situation for the SADatabase agent.

To test SADatabase agent failover

1. Connect to the database from Interactive SQL. For example:

```
dbisql -c "UID=DBA;PWD=sql;Server=VCS;LINKS=tcpip"
```

2. Execute the following query:

   ```
   SELECT * FROM Departments;
   ```

   The query should execute without errors.

3. Suppose the database failed, and the database server running on the first system node cannot access the database file. This would create a failover of the database file to the database server started on the second system node. You can cause the database file on the first node to fail by issuing a command similar to the following:

   ```
   dbisql -q -c "UID=DBA;PWD=sql;Server=VCS1;DBN=utility_db" STOP DATABASE
   DEMO ON VCS1 UNCONDITIONALLY;
   ```

   The database file on the first computer fails. There is a delay before Veritas Cluster Server recognizes that the file has failed because Veritas Cluster Server monitors the health of its resource, every 60 seconds by default (you can make this interval smaller in your resource configuration). The database file then fails over to the second computer, and that database file will be started using the database server on the second computer, which may have a different name than the original database server.

   For example, if the new database server is called VCS2, then clients must specify the new database server name in their connection strings:

   ```
   "UID=DBA;PWD=sql;Server=VCS2;DBN=DEMO;LINKS=tcpip"
   ```

4. Reconnect from Interactive SQL. You should be able to connect and execute the query successfully.

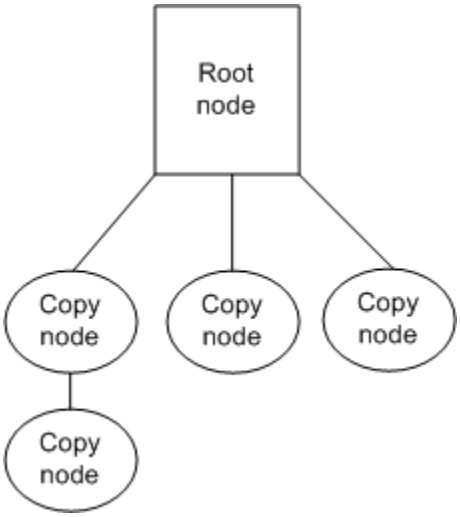
SQL Anywhere read-only scale-out

Separately licensed component required
Read-only scale-out requires a separate license. See “Separately licensed components” [*SQL Anywhere 12 - Introduction*].

Read-only scale-out is a configuration that allows you to offload reporting or other operations that require read-only access to the database.

A scale-out tree consists of the root node and copy nodes. The **root node** of a scale-out system has the only copy of the database that is writable. The root node can be either a single database, or a database mirroring system. Below the root node, there can be one or many branches of copy nodes. A **copy node** is a database server that runs a copy of a database for read-only access, and a copy node can be the parent of other copy nodes.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0



How scale-out works

The root node is the only database server that accepts both read and write requests. Once you start additional copy nodes, the root database server sends transaction log pages to the copy nodes in the tree that are defined as its children, provided that they are connected and ready to receive transaction log pages. The pages are normally sent without waiting for a response; however, the root database server occasionally requests an acknowledgement to ensure that the copy node does not receive more asynchronous requests than it can handle.

When the copy node receives pages, it writes them to disk and then sends them to its children (if it has any).

The parent detects if a child node becomes unavailable, and if this happens, the parent stops pushing transaction log pages to the child. If the child is restarted, it requests the transaction log pages that it does not have, and then the parent resumes pushing transaction log pages to the child. The child notifies its parent of changes in the status of the copy node, and the status information eventually makes its way up through the tree to the root database server.

If the root database server becomes unavailable, all of the children in the scale-out system continue running, but they no longer receive updates from the primary database server. Any connections to the copy nodes may retrieve data that is stale. When the root database server becomes available again, its children re-establish connections and resume receiving transaction log pages.

You can use scale-out in conjunction with database mirroring if you want to ensure the availability of your database. See “Using read-only scale-out with database mirroring” on page 989.

Connecting to a database in a scale-out system

To set up a scale-out system, it is recommended that you create two definitions with the CREATE MIRROR SERVER statement for the database server that is acting as the root node: one definition for the partner role and one definition for the primary role. In a scale-out system, you must define one partner server. The name that is specified in the partner server definition is used in connection strings and when starting the database server. The database server defined as the primary server is the default parent for

copy nodes in the scale-out system. If there is no primary server defined, then you must specify the parent for each copy node that is added to the system.

When you use read-only scale-out, the application connects to the root database server, and the root database server uses information from the application's connection string, together with status and load information from the copy nodes, to determine which node the application should connect to. You can choose to have the application connect to the copy node that is least heavily loaded, so the root database server redirects the client to that node. If an application makes and drops several such connections within a short period of time, the connection is pooled and the root database server is not asked which copy server to use. This behavior reduces the load on the root database server, but may not give expected behavior. The application can specify that its connections are not to be pooled to ensure that the root server determines which copy node to connect to on each connection. See "Connection pooling and read-only scale-out" on page 137.

Tip
You can check the status of the database servers in a scale-out system by connecting to the primary database from Sybase Central. This information is available on the **Health And Statistics** pane. See "Monitoring database health and statistics" on page 695.

Tutorial: Setting up a read-only scale-out system

This tutorial takes you through the steps of setting up a root database server that automatically adds a child node.

Note
There is a sample in *samples-dir\SQLAnywhere\DBMirror* that uses a database mirroring system in conjunction with a scale-out system.

For information about the location of *samples-dir*, see "File locations and installation settings" on page 391.

To set up a read-only scale-out system

1. Create the following directory: *c:\scaleoutdemo*.

2. Run the following command to create *scaleoutdemo.db* that contains data from the sample database:

```
newdemo c:\scaleoutdemo\scaleoutdemo.db
```

3. Start the root database server for the scale-out system:

```
dbsrv12 -n scaleout_root_demo -su sql -x TCPIP(port=6871) "c:\scaleoutdemo
\scaleoutdemo.db" -xp on
```

4. Connect to the root database from Interactive SQL:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=scaleout_root_demo;DBN=scaleoutdemo"
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

5. Define the root database server for the scale-out system:

```
BEGIN
    EXECUTE IMMEDIATE
    'CREATE MIRROR SERVER "scaleout_primary_demo"
    AS PRIMARY
        connection_string = ''SERVER=scaleout_primary_demo;HOST='
        || PROPERTY( 'MachineName' ) || ':6871''';
END
```

6. Define the root database server as a partner in the scale-out system. The name of the partner server must match the database server name that is used in the command to start the database server.

```
BEGIN
    EXECUTE IMMEDIATE
    'CREATE MIRROR SERVER "scaleout_root_demo"
        AS PARTNER
        connection_string = ''SERVER=scaleout_root_demo;HOST='
        || PROPERTY( 'MachineName' ) || ':6871''';
END
```

7. Set the options for the root server for the scale-out system:

```
SET MIRROR OPTION auto_add_server='scaleout_root_demo';
SET MIRROR OPTION child_creation='automatic';
SET MIRROR OPTION authentication_string='abc';
SET MIRROR OPTION auto_add_fan_out='10';
```

8. Make a backup copy of the database, placing it in the directory *c:\scaleoutdemo\copynode*.

```
BACKUP DATABASE DIRECTORY 'c:\\scaleoutdemo\\copynode';
```

9. Start the backup copy of the database as a child (copy node) of the scaleout_root_demo database server:

```
dbsrv12 -n scaleout_child_demo -su sql -x TCPIP(port=6873) "c:\scaleoutdemo
\copynode\scaleoutdemo.db" -xp on
```

10. Connect to the child node from Interactive SQL:

```
dbisql -c "UID=DBA;PWD=sql;SERVER=scaleout_child_demo"
```

    Once the child node connects to its parent, you are warned that the child is a read-only copy of the database. You can now connect to the copy node and execute queries against the copy of the database.

11. You can view the mirror servers in the scale-out system by running the following query:

```
SELECT * FROM SYSMIRRORSERVER;
```

12. Disconnect from Interactive SQL and shut down the database servers.

13. (optional) Delete the *c:\scaleoutdemo* directory.

Setting up read-only scale-out

The following steps must be performed to configure a scale-out system:

1. Start a database server with the **-xp on** database option. This database server is the root node for the scale-out system.

2. Add the scale-out object definitions to the root database. You must define the servers and set options for the servers.

3. Make backup copies of the root database.

4. Start the copy nodes (the backup copies of the database).

Set up the root node

When you configure the root node, define the root database server as a partner server. You can also define a primary server.

To set up the root node

1. Start a database server running the database that you want to have read-only copies of. You must specify the **-xp on** database option when starting the database server.

2. Define the scale-out objects in the root database using the CREATE MIRROR SERVER statement.

   It is recommended that you create two definitions for the root database server: CREATE MIRROR SERVER ... AS PARTNER and CREATE MIRROR SERVER ... AS PRIMARY. The name that you give the database server in the statement that uses the AS PARTNER server is the name that is used in the command to start the database server and in client connection strings. The name that you give the database server in the statement that uses the AS PRIMARY clause is the name of the database server that is the default parent for copy nodes that are added to the scale-out system. If you do not define a primary server, then you must specify the name of the parent server when you create copy nodes, so defining a primary server is recommended.

3. Set scale-out options for the database using the SET MIRROR OPTION statement.

See also

- "-xp dbsrv12 database option" on page 263
- "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]
- "SET MIRROR OPTION statement" [*SQL Anywhere Server - SQL Reference*]
- "Adding copy nodes" on page 984

Adding copy nodes

The child_creation option of the SET MIRROR OPTION statement controls how child nodes are added to a read-only scale-out system. The following values are supported:

- **Automatic** The root database server authenticates the copy node once it starts, or creates a new copy node if the copy node is not known. This is the recommended setting because the root server creates the definition for unknown copy nodes so that you do not have to create them manually.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Off** You must connect to the root database server and execute a CREATE MIRROR SERVER statement to add a new copy node.

- **Manual** You can add copy nodes to the tree by connecting to a copy node and executing a CREATE MIRROR SERVER statement for that database server. This value requires DBA authority. The copy server sends a request to the root database server to define the new copy node. Once the copy node is defined, the root database server allows the new copy node to request log pages.

The database stores the connection string that is associated with the primary database server in the system. When you start a new copy node that has not been previously defined, it connects to the root database server using this connection string. The root database server uses the value of the authentication_string option that is stored in the database to authenticate the copy node.

As part of the mirror connection request, the copy node sends the copy database server's name and a string containing the copy database server's IP address(es) and port(s) to the root database server. Once the copy node is authenticated, the root database server determines whether the copy node is known. If the copy node is not known, then the root database server executes a CREATE MIRROR SERVER statement to define the new copy and its connection string. Once the copy is known, the root database server can establish a connection to it. The copy then requests all the transaction log pages that it does not already have, and once the copy node has them, the root database server starts pushing new transaction log pages to the copy node.

See also

- "SET MIRROR OPTION statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]

Define copy nodes

The definitions for the copy nodes are stored in the database. You can choose to define copy nodes for the scale-out system in advance or to have the root database server define the copy nodes when they connect. It is recommended that you let the root database server create the copy node definitions because this process reduces the possibility of errors occurring in your copy node definitions.

To define copy nodes automatically

1. Make backup copies of the root database.

2. Start the copies of the database as copy nodes. You must specify the **-xp on** database option when starting the database server.

See also

- "-xp dbsrv12 database option" on page 263

Assigning a parent to a copy node

A copy node must have a parent. The parent can be the root database server, or another copy node. You can set up the read-only scale-out system so that the parent is assigned automatically. You can also change the position of a node within the scale-out hierarchy at any time.

Automatically assign the parent of a copy node

The SET MIRROR OPTION statement supports two options that can be used to assign new copy nodes to a parent in the tree so that the work of distributing transaction log pages is balanced among the nodes:

- **auto_add_server** Specifies the name of a database server that acts as the top-most node of the automatic assignment tree.
- **auto_add_fan_out** Specifies the maximum number of copy nodes that each branch in the tree should have. The default is 10. New nodes that are created as a result of one of the following actions have their parent assigned automatically:
  - connecting to the primary server when the child_creation option is set to automatic
  - executing CREATE MIRROR SERVER ... USING AUTO PARENT on the copy node when the child_creation option is set to manual
  - executing CREATE MIRROR SERVER ... USING AUTO PARENT on the root database server

If the auto_add_server option is defined, the database server specified by that option is used as the top-most node; otherwise, the root database server is used as the top-most node of the automatic assignment tree. The root database server determines which node in the tree should be assigned as the parent for the new copy node, based on the number of children for each node and its status as reported to the root database server by periodic status updates. The copy node sees the assignment of its new parent as it applies transaction log operations received from the root database server, so the copy node changes its parent connection when it receives the transaction log operation.

See also

- "SET MIRROR OPTION statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]

Changing the position of a node

Once a copy node is running, you can assign a new parent to it by using the ALTER MIRROR SERVER statement on the root database server:

```
ALTER MIRROR SERVER "copy-server-name"
 FROM SERVER "new-parent-name";
```

The statement is recorded in the transaction log on the root database server. When the change is pushed to the copy node, the node recognizes that its own definition is being altered, and then connects to the new parent that is specified in the statement.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "ALTER MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]

Determining the parent of a copy node

A copy node determines what server to connect to by looking in the ISYSMIRRORSERVER system table for its own mirror server definition and seeing what its parent value is.

- Null means the copy node's parent is the root node.
- If there is an ID in the parent column, then the server with that ID is the copy node's parent.

Once the copy node knows which server is its parent, it uses the connection_string option value stored in the database to connect to the correct parent.

Determining the parent of a copy node in a database mirroring system

If scale-out is used together with a database mirroring system, copy nodes can be defined so that their partner is the server currently acting as the primary server or as the mirror server. In this configuration, one server is defined in ISYSMIRRORSERVER with type PRIMARY and one with type MIRROR. Either of these servers can be selected as a parent for a copy node.

See also

- "SYSMIRRORSERVER system view" [*SQL Anywhere Server - SQL Reference*]
- "SET MIRROR OPTION statement" [*SQL Anywhere Server - SQL Reference*]

Handling the loss of a parent connection

When the parent server for a copy node becomes unavailable, the copy node's database remains available. The copy node continues to try to connect to its parent for a period defined by the max_retry_connect_time mirror option (by default, 120 seconds). If a connection cannot be established in the specified time, the copy attempts to connect to its alternate parent (defined using the OR SERVER clause of the CREATE MIRROR SERVER STATEMENT) if one was defined. If an alternate parent was defined, the copy attempts to connect to it for an additional max_retry_connect_time seconds. If a connection cannot be made, the copy tries to connect to the root database server and requests that its former parent be replaced with the copy node itself, assigning any of the copy node's siblings as its children. This behavior can result in a situation where the copy node has more children than specified by the auto_add_fan_out setting.

If the copy server is unable to establish a connection to the root database server within max_disconnected_time seconds (default unlimited) after the parent connection was lost, the database is shut down.

During the time the copy server is trying to connect to another database server, it continues to try to connect to its original parent. If that connection attempt is successful, obtaining log pages from the original parent database server is resumed.

When a read-only node is first started, it makes its database available for read-only connections even if other nodes in the tree, including its parent, are not available.

Determining the state of a server in a scale-out system

You can use the following statement to determine the state of a connection to a copy node's parent:

```
SELECT DB_EXTENDED_PROPERTY('MirrorServerState',server-name );
```

You can get an indication of how much of the transaction log has been processed on any of the nodes by using the following statement:

```
SELECT DB_PROPERTY('CurrentRedoPos');
```

The value of the CurrentRedoPos property on a child node can be compared to the same property on the primary server to get an indication of how up-to-date the copy server is.

The sa_mirror_server_status system procedure returns the connection status of all servers below the server on which the procedure is executed. See "sa_mirror_server_status system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "SET MIRROR OPTION statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]

Connecting to copy nodes

You can connect to a copy node by connecting to the primary server and specifying the NodeType connection parameter as part of the connection string. The NodeType connection parameter accepts the following values:

- **COPY** When the NodeType connection parameter is set to COPY, the database server examines the copy nodes in its own branch (including itself if it is not the root node) and chooses the copy node with the lightest load. If the database server does not choose itself, it redirects the client to the chosen database server.
- **DIRECT** This is the default setting. When the NodeType connection parameter is set to DIRECT, the database server accepts the connection without performing load balancing or redirection.
- **PRIMARY** If the NodeType connection parameter is set to PRIMARY and you have connected to the primary server, the connection is accepted. If you have connected to the mirror server or a copy node, the database server redirects the connection to the primary server.

Once the root database server determines which copy node should be used, the database server returns a connection string to the client containing the necessary information to connect to that node. The client then makes a connection to the copy node and communicates directly with its new parent database server for all further requests. The root database server determines which copy node to select as the parent based on status information that is sent to the root database server periodically by the copy nodes.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "NodeType (NODE) connection parameter" on page 300

Dropping servers from a scale-out system

You can use a DROP MIRROR SERVER statement to drop servers in a read-only scale-out system.

When you set up a scale-out system, the root database server may have two definitions (CREATE MIRROR SERVER ... AS PARTNER and CREATE MIRROR SERVER ... AS PRIMARY). You may need to drop both the role and partner server definitions from the database, depending on your reason for dropping the root server.

Note
If you want to keep the same server name but change its settings, you can use the CREATE OR REPLACE MIRROR SERVER statement or the ALTER MIRROR SERVER statement. See "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*] and "ALTER MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*].

To drop a mirror server (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a DROP MIRROR SERVER statement to drop the mirror server. For example:

```
DROP MIRROR SERVER mirror-server-name;
```

See also

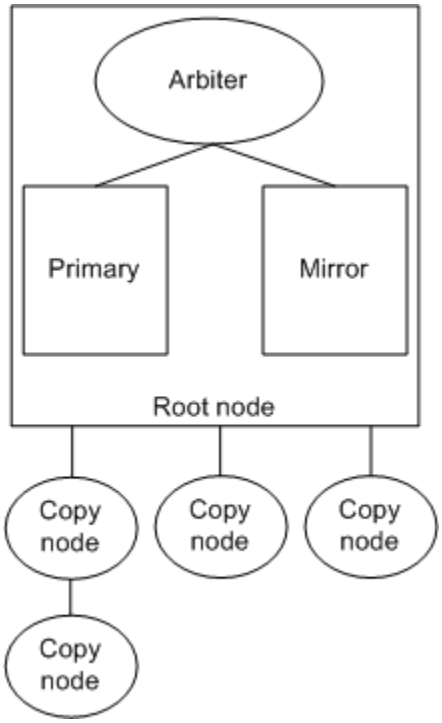
- "DROP MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]

Using read-only scale-out with database mirroring

For information about using high availability and/or read-only scale-out for hosting web applications, see "SQL Anywhere web services high availability and scale-out solutions" on page 990.

In a database mirroring system, you can access the database running on the mirror server using a read-only connection. This functionality is useful if you want to offload reporting or other operations that require read-only access to this database. The only difference between a mirror server and a copy node is that a copy node cannot participate as a primary or mirror server in a database mirroring system.

You can use database mirroring with read-only scale-out to ensure the availability of the root node. When scale-out is used with database mirroring, the root node of the scale-out system consists logically of the primary, mirror, and arbiter servers, instead of a single database server.



Once you start the database servers in the mirroring system, additional database servers can access the participating mirror servers to maintain read-only copies of the database. You can add as many branches of copy nodes to the mirror servers as necessary.

Differences between copy nodes and primary and mirror servers

When copy nodes are used in a database mirroring system, they can be the child of the current primary server, the current mirror server, or another copy node. The copy node gets its transaction log pages from its parent. Unlike mirror nodes, copy nodes do not have a state information file because their state does not influence which database server has ownership of the database.

See also

- "Introduction to database mirroring" on page 945
- "Configuring read-only access to a database running on the mirror server" on page 967
- "ISYSMIRRORSERVER system table" [*SQL Anywhere Server - SQL Reference*]
- "CREATE MIRROR SERVER statement" [*SQL Anywhere Server - SQL Reference*]

SQL Anywhere web services high availability and scale-out solutions

The following SQL Anywhere components are required to configure high availability and/or scale-out solutions for hosting web applications:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Relay Server (RS)** The Relay Server is the entry point for HTTP requests (typically from the Internet), providing virtual hosting and load balancing capability. The Relay Server accepts and routes HTTP requests that target virtual entities called a **server farm**. Each server farm may consist of one or more SQL Anywhere database/web servers. The Relay Server is the HTTP/WEB access point that relays all HTTP requests to an appropriate back-end server for processing. See "Introduction to the Relay Server" [*Relay Server*].
- **Outbound Enabler (OE)** The Outbound Enabler provides a bridge between the Relay Server and a back-end database server. There is one Outbound Enabler process for each back-end server residing within a server farm. Its purpose is to ensure that its designated back-end server is operational, and it is responsible for establishing a communication channel with the Relay Server and forwarding requests to its back-end server. Since the Outbound Enabler initiates the connection to the Relay Server, it can easily make an outbound connection from a LAN, through a firewall, to a Relay Server situated in the DMZ. See "Outbound Enabler" [*Relay Server*].
- **SQL Anywhere (back-end) server** SQL Anywhere (back-end) server is an advanced database and web server that can host web applications written entirely in SQL. SQL Anywhere servers may be deployed to mitigate demanding environments requiring high availability and scale-out solutions. High availability provides fault tolerance capabilities to protect against interruption of service and scale-out provides load balancing to increase throughput during peak loads.

High availability

Three SQL Anywhere servers are required to provide high-availability: an **arbiter** and two **partner** servers. One partner is assigned the role of the **primary** and the other is the **mirror**. Write operations, such as an insert to a global table, are only permitted on the primary, while read operations, such as a select, are permitted on both types of servers. The primary automatically keeps its mirror synchronized as it receives updates. In the event of a failure on the primary, the arbiter switches the roles of the partners such that the mirror becomes the primary.

High availability may be leveraged within a web service environment by routing all HTTP requests through a Relay Server. The Relay Server processes the URL of the HTTP request to derive the target server farm. For simplicity, consider a Relay Server configuration that routes HTTP requests to either a readFarm or a writeFarm. Although the primary back-end server may process both read and write requests, you can configure it to process write requests only if there are other read-only copy nodes to cover the read functionality in case of server failure; otherwise, you can configure it to process both read and write requests. On the other hand, you may configure the mirror server to process read requests to spread the read load, or it can be configured not to process any requests to reduce hardware use on the mirror server. The client application requires no knowledge of the physical server that it will access for read and write requests; it simply composes its URL to direct the request to either the readFarm or the writeFarm.

In addition to the Relay Server, each of the partnered back-end servers requires two Outbound Enablers, one for each farm the back-end server participates in. Two Outbound Enablers control the communication channels to the primary partner and two Outbound Enablers establish communications with the mirror partner. Within this configuration, at any instant in time there is only one primary server and it activates its Outbound Enabler assigned to the writeFarm, and optionally deactivates its Outbound Enabler associated with the readFarm if there are other members in the readFarm, other than the copy nodes. The mirror deactivates its Outbound Enabler associated with the writeFarm and optionally activates its Outbound Enabler associated with the readFarm.

If the primary server becomes unavailable because of hardware or software failure, the mirror server negotiates with the arbiter to assume the role of primary server, and thus become a member of the writeFarm. It should also become a member of the readFarm if the system has no other read-only copy nodes. At the time of failure, the original primary is abruptly withdrawn from both the writeFarm and the readFarm if it is a member. The original primary server may not recover until the cause of the failure has been resolved. When it recovers, it assumes the mirror role, and optionally joins the readFarm. The abrupt failure may fail requests currently being processed by the failing server. The latency of writeFarm recovery from the mirror depends on the role-switch latency of the SQL Anywhere mirror plus the status polling interval specified on the writeFarm Outbound Enabler of the mirror. This status polling interval can be set and its default is 5 seconds if not explicitly specified.

Scale-out

A minimum of two SQL Anywhere servers are required to provide scale-out capability: a root node and a copy node. A copy node, like a mirror server, is read-only, but it differs in that it can never assume the primary role. It is strictly intended to service read requests. As discussed in the high availability scenario, the primary is associated with the writeFarm and the copy node replaces the mirror server as the readFarm member. With one root node and a single copy node, only two Outbound Enablers are required in total, one for each server. Optionally, you may add the primary to the readFarm so that your environment remains fully functional when all the copy nodes are down. The Outbound Enabler associated with the root node is configured to receive requests to the writeFarm while the copy node's Outbound Enabler is configured to accept requests to the readFarm. The root node continually updates its copy node as it receives write updates. The configuration can be expanded by adding copy nodes to the readFarm. Running 10 copy nodes to the readFarm provides load balancing and mitigates server failures incurred within the readFarm.

High-availability plus scale-out

The above configuration scenarios may be combined to provide both high-availability and scale-out. Such a configuration would provide fault tolerance for writable (primary/mirror) and readable (mirror and copy) back-end servers. In addition, adding more copy nodes increases the throughput potential for HTTP read-only requests. In this configuration, the primary and mirror server make up the root node, and the primary, the mirror, and the copy nodes all have one Outbound Enabler for participation in the readFarm. The primary and mirror have one additional Outbound Enabler for participation in the writeFarm. Optionally, you can remove the primary and mirror from the readFarm.

OE configuration

For the purposes of this discussion, an Outbound Enabler must be configured to identify the following:

- its associated server farm
- Relay Server connection parameters
- SQL Anywhere (back-end server) connection parameters

For a detailed explanation of Outbound Enabler command line options, see "Outbound Enabler syntax" [*Relay Server*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The following example Outbound Enabler command line sets up a connection to the Relay Server based on parameters provided by the -cr option. If it is active, the Relay Server forwards requests made to the readFarm to the backend server configured according to the -cs option.

```
rsoe.exe -f readFarm -id saHost1 -cr url_suffix="/ias_relay_server/server/
rs_server.dll";host=rs_server;port=80 -cs host=sahost1;port=8080;status_url=/
satest/oe_read_status?ro=1
```

Based on the above example, a request URL of the form `http://rs_server/ias_relay_server/client/rs_client.dll/readFarm/service_path[?service_query]`

directs the request to the Relay Server hosted on rs_server. The Relay Server strips off the domain and farm components from the URL and relays the request to an active Outbound Enabler associated with the farm, over the RS-OE channel. The Outbound Enabler then forwards the request to its backend server.

An Outbound Enabler maintains a channel to the Relay Server only if it detects that the backend server is ready to process requests. It does this by polling the server at a configurable interval, sending an HTTP request specified by the -cs option. The status_url option provides the specific URL path for the ping service. In the example configuration above, the Outbound Enabler sends HTTP requests to `http://sahost1:8080/satest/oe_read_status?ro=1`. If the server responds with **AVAILABLE=TRUE | ON | 1** then the Outbound Enabler maintains or initiates a communication channel with the Relay Server hosted on rs_server.

SQL Anywhere configuration

For details on how to configure and start SQL Anywhere as a web server, see "Using SQL Anywhere as an HTTP web server" [*SQL Anywhere Server - Programming*].

Leveraging the rsoe (Relay Server outbound Enabler) startup example discussed in the Outbound Enabler configuration section, an oe_read_status service may be written as follows:

```
// READ-ONLY SA-OE ping service
//  optionally includes primary as a READ node.
call sa_make_object( 'service', 'oe_read_status' );
alter service oe_read_status
    type 'raw'
    authorization off
    secure off
    user DBA
    as call sp_oe_read_status(:ro);

// if ro is 1 then only read-only servers are activated: mirror or copy
// if ro is 0 then include all servers: primary, mirror, copy...
create or replace procedure sp_oe_read_status( ro int )
begin
    declare readonly long varchar;
    declare res long varchar;

    set res = 'AVAILABLE=';

    set ro = isnull(ro, 0);
    if ro = 1 then
         select db_property('ReadOnly') into readonly;
          if readonly = 'On' then
               set res = res || 'TRUE';
          else
               set res = res || 'FALSE';
```

```
            end if;
        end if;;

        call sa_set_http_header('Content-Length', length(res) );
        select res;
    end;
```

The above web service and stored procedure could be used for any SQL Anywhere server that has the role of a primary, mirror, or copy node. If the ReadOnly database property is set to On, then **AVAILABLE=TRUE** is returned. With minor modifications, a service for use within a writable server farm, for example writeFarm, would return **AVAILABLE=TRUE** only when ReadOnly is set to Off. Optionally, a ping service for an Outbound Enabler may base its AVAILABILITY on other factors such as an entry in a global table. For example, a staged deployment might update a table or procedure on the primary which, when propagated to the mirror and copy nodes, activates or deactivates each server's server farm membership.

Relay server sonfiguration

To configure the Relay Server:

1. Create the readFarm and writeFarm backend farms.
2. Configure the primary, mirror, and copy nodes to be members of the readFarm Relay Server backend farm.
3. Configure the primary and mirror to be members of the writeFarm Relay Server backend farm.
4. For both farms, set the active_cookie configuration option to **yes** and the active_header configuration option to **no**. (This last step can be done in Sybase Central by selecting **SQL Anywhere** as the backend farm affinity type.

See:

- "Relay Server configuration file" [*Relay Server*]
- "Updating a Relay Server farm configuration" [*Relay Server*]
- "Backend farm section properties" [*Relay Server*]

Client/application configuration

- **Limitations** Currently, SQL Anywhere back-end servers can only support high availability running a single database. The maximum latency for high availability fail-over is the maximum of:
  - The polling frequency of the Outbound Enabler. See the -d option in "Outbound Enabler syntax" [*Relay Server*].
  - The time it takes for the mirror to become the primary.

Troubleshooting SQL Anywhere database issues

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Frequently asked questions - SQL Anywhere" [*SQL Anywhere 12 - Introduction*]
- "Performance improvement tips" [*SQL Anywhere Server - SQL Usage*]
- "Query optimization and execution" [*SQL Anywhere Server - SQL Usage*]
- "Application profiling" [*SQL Anywhere Server - SQL Usage*]
- "SQL Anywhere Monitor" on page 1007
- "Using the SQL Anywhere Console utility" on page 759
- "Monitoring and improving database performance" [*SQL Anywhere Server - SQL Usage*]

Understanding unexpected changes in the database size

Database pages are freed when all records on the page are deleted. When a database page is freed, it becomes available for reuse, but it cannot be removed from the file. However, future INSERT and UPDATE statements can use the freed pages.

Whenever modifications such as inserts, updates, or deletes are made to the database, entries are added to the rollback log, which is stored within the system dbspace. If many of these operations are performed before a commit is executed, the rollback log can become very large and may increase the size of the database.

The checkpoint log is stored at the end of the system dbspace. When the database server shuts down, the checkpoint log is truncated and the system dbspace shrinks. Pages that were freed by DELETE or TRUNCATE operations remain within the database file for future reuse and cannot be removed from the file.

If the size of your database file is increasing, or is not decreasing as expected:

- Execute COMMITs frequently if you are using INSERT, UPDATE, or DELETE statements. Pages allocated for the rollback log are freed for reuse in the system dbspace when a COMMIT is performed.

- Execute CHECKPOINTs occasionally when you are using UPDATE or DELETE statements, or if you are using INSERT statements and large indexes are involved. Pages in the checkpoint log become available for reuse by the checkpoint log after each checkpoint.

- Execute TRUNCATE TABLE, which can result in page-level deletes. In these cases, copies of the pages do not need to be added to the checkpoint log and individual row-level operations do not need to be added to the rollback log. Pages freed by TRUNCATE TABLE are only reusable after the next checkpoint. TRUNCATE TABLE results in page-level deletes when the following conditions are true:

    - There are no foreign keys to, or from, the table being truncated.
    - TRUNCATE TABLE is not being executed within a trigger.
    - TRUNCATE TABLE is not being executed in conjunction with an atomic operation.
    - The checkpoint log pages are written to the end of the system dbspace file. These pages are removed when the database is shut down.

Rebuilding the database can decrease the size of the database because the rebuilt database has fewer free pages.

Error reporting in SQL Anywhere

When a fatal error or crash occurs and is detected by any of the following applications, an error report is created about what was happening at the time of the problem:

- Interactive SQL (dbisql)
- MobiLink Listener (dblsn)
- MobiLink server (mlsrv12)
- network server (dbsrv12)
- personal server (dbeng12)
- QAnywhere agent (qaagent)
- SQL Anywhere client for MobiLink (dbmlsync)
- SQL Anywhere Console utility (dbconsole)
- SQL Remote (dbremote)
- Sybase Central

The error report includes information such as the execution state of the threads at the time of the crash, so that iAnywhere Solutions is better able to diagnose the cause of the problem. By default, the error report is created in the diagnostic directory (specified by the SADIAGDIR environment variable), or if this location does not exist, it is created in the same directory as the database file.

Error report file names are composed as follows:

- a prefix that identifies the application:

Application identifier	Application
LSN	Listener utility
MLC	MobiLink client
MLS	MobiLink server
QAA	QAnywhere agent
SA	Personal or network database server
SR	SQL Remote

- a value indicating the software version
- two fields linked with underscores that provide the timestamp for when the error report was created
- the application identifier
- the extension *.mini_core*

For example, *SA12_20090620_133828_32116.mini_core* is an error report from a SQL Anywhere version 12 database server from 2009/06/20, at 1:38:28 p.m., from process 32116.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

During normal database server operation, diagnostic information is also recorded about the database server, such as how many CPUs are on the computer, whether hyperthreading is enabled, and what options were specified when the server was started. This information can also be submitted using dbsupport.

How SQL Anywhere software submits error reports and diagnostic information

After the database server successfully writes out error report information, it launches the Support utility (dbsupport) and passes it the name of the error report file to be submitted. By default, dbsupport attempts to prompt you to submit an error report when it is generated, but if dbsupport is unable to prompt you, then the report is not sent. iAnywhere encourages you to submit error reports when they occur. The report does not contain any information that identifies the sender.

Error reports and diagnostic information are uploaded to the iAnywhere Error Reporting web site via HTTP. This process saves you time by making it as convenient as possible to send relevant files to iAnywhere so that it is possible to diagnose and provide solutions to problems you encounter.

You can change the default behavior of dbsupport with the -cc option:

- The following command configures dbsupport to submit error reports automatically without prompting the user:

  ```
  dbsupport -cc autosubmit
  ```

- The following command turns off automatic error report submission:

  ```
  dbsupport -cc no
  ```

If you choose not to submit an error report, it remains in the diagnostic directory on your hard disk. The location of the diagnostic directory is specified by the SADIAGDIR environment variable. See "SADIAGDIR environment variable" on page 382.

You can view the list of error reports with the -lc option:

- The following command generate a list of all crash reports that have not been submitted to iAnywhere Solutions:

  ```
  dbsupport -lc
  ```

Submitting error reports to iAnywhere Solutions assists with diagnosing the cause of a fatal error or assertion. Once an error report is submitted, it is deleted from the computer where it was generated.

You can manually submit the error reports with the -sa, -sc or -sd option:

- The following command submits all crash report and diagnostic information stored in the diagnostic directory to iAnywhere Solutions:

  ```
  dbsupport -sa
  ```

You can also include comments and files with the error report by specifying the -ac and -af options, respectively.

```
dbsupport -sc SA12_20080901_113308_3360 -ac "The message.txt file provides
more
information about this error report." -af c:\scenario.txt -af c:\message.txt
```

See also

- "Support utility (dbsupport)" on page 853

Troubleshooting database server startup

This section describes some common problems that may occur when starting the database server.

Ensure that your transaction log file is valid

The database server won't start if the existing transaction log is invalid. For example, during development you may replace a database file with a new version, without deleting the transaction log at the same time. However, doing so causes the transaction log file to be different than the database, and results in an invalid transaction log file.

Ensure that you have enough disk space for your temporary file

SQL Anywhere uses a temporary file to store information while running. This file is usually stored in the directory pointed to by the SATMP environment variable, typically *c:\temp*.

If you do not have enough disk space available to the temporary directory, you will have problems starting the server.

See "SATMP environment variable" on page 385.

Ensure that network communication software is running

Appropriate network communication software must be installed and running before you run the database server. If you are running reliable network software with just one network installed, this process should be straightforward.

For more information about network communication issues, see "Client/server communications" on page 76.

You should confirm that other software requiring network communications is working properly before running the database server.

If you are running under the TCP/IP protocol, you may want to confirm that ping and Telnet are working properly. The ping and Telnet applications are provided with many TCP/IP protocol stacks.

Debugging network communications startup problems

If you are having problems establishing a connection across a network, you can use debugging options at both the client and the server to diagnose problems. On the database server, you use the -z option. The

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

startup information appears in the database server messages window: you can use the -o option to log the results to an output file.

See "-z dbeng12/dbsrv12 server option" on page 243 and "-o dbeng12/dbsrv12 server option" on page 208.

Create a debug log file

You can use the LogFile connection parameter to create a debug log file. Log files can provide more details about where a connection failure occurred, thereby helping you troubleshoot and correct the problem. See "LogFile (LOG) connection parameter" on page 298.

Using the correct *sasrv.ini* file

If you are having problems establishing a connection to the correct server across a network, try deleting the *sasrv.ini* file. This file contains database server information, including the database server name, protocol, and address. The server information in this file could be overriding information you specified in the connection string. Deleting this file causes SQL Anywhere to create a new *sasrv.ini* file containing the information you specify in the connection string.

Operating system	Default location of *sasrv.ini*
Windows	*%ALLUSERSPROFILE%\Application Data\SQL Anywhere 12*
Windows Mobile	The same directory as the executable that is running
Unix	*$HOME/.sqlanywhere12*

If you continue to experience problems establishing a connection, you should also delete any copy of *sasrv.ini* located in any of the following places:

- The *bin32* or *bin64* subdirectory of your SQL Anywhere installation directory (listed in the *HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\SQL Anywhere\12.0\Location* registry key)
- Windows directory
- Windows system directory
- Anywhere else in your path

For more information about the *sasrv.ini* file, see "Server name caching for faster connections" on page 144.

Understanding database server performance warnings

Database server performance warnings appear in the database server message log. These warnings are reported when a database or database server reaches a state where its performance may be affected.

Performance warning: Server cache size is too small for database %1

This warning is reported if the database server's maximum cache size (the upper limit of the database server's buffer pool size, as computed by the server or as explicitly specified by the -ch option) is less than 10% of the total size of all the open dbspaces for a database.

For example, a database server hosting a 21 GB database would issue this warning if the server's maximum cache size was less than 2.1 GB. This warning is computed on a per-database basis, and may not be issued even if multiple databases are started on the same database server.

See also

- "-ch dbeng12/dbsrv12 server option" on page 163
- "Use the cache to improve performance" [*SQL Anywhere Server - SQL Usage*]
- "Dynamic cache sizing" [*SQL Anywhere Server - SQL Usage*]
- "Monitoring cache size" [*SQL Anywhere Server - SQL Usage*]

Performance warning: Database %3 has a page size of %1 that does not match maximum of %2 set for server, causing inefficient use of cache"

The page size of the database server's buffer pool must be at least as large as the page size of any database started on that database server. A database server can use a larger page size for the cache, which can be specified using the -gp option. However, in a SQL Anywhere database server, there must be a 1:1 mapping between database pages and page frames in the buffer pool. Each page frame in memory can contain only one database page, and any page may appear in only one page frame. If the cache uses 8 KB pages, but the database uses 4 KB pages, then each cache page frame in memory contains only 4 KB of data, and the other 4 KB of that page frame are wasted. This situation results in a cache size of half the desired amount.

See also

- "-gp dbeng12/dbsrv12 server option" on page 193
- "Setting a maximum page size" on page 39

Performance warning: No unique index or primary key for table %1 in database %2

This warning is reported when a table is updated that contains more than 10 rows and that does not have a primary key or a unique index.

In SQL Anywhere, the transaction log contains logical row operations (UPDATE, INSERT, and DELETE) rather than the changes to physical pages. Row operations in the transaction log are identified by primary key or unique index value, so that if you recover a database, the updated values for that

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

operation in the transaction log can be applied to the correct row. If no primary key or unique index exists, then all of the values in the row are used as the row's primary key. This behavior can result in significant transaction log growth. A table that does not have an index may result in slow database recovery times.

See also

- "Controlling transaction log size" on page 24
- "Working with indexes" [*SQL Anywhere Server - SQL Usage*]
- "The transaction log" on page 21

Performance warning: Page size too small for database %1

This performance warning is reported when a database meets all of the following conditions:

1. The database page size is 2 KB.
2. The number of pages in the entire database is greater than 100000, corresponding to a database size of approximately 200 MB.

It is recommended that you unload the database and then reload it into a new database with a 4 KB or larger page size for greater space efficiency and better performance.

See also

- "Rebuilding databases" [*SQL Anywhere Server - SQL Usage*]
- "Table and page sizes" [*SQL Anywhere Server - SQL Usage*]

Performance warning: Database file %1 consists of %2 disk fragments

When a database starts running on a database server, SQL Anywhere reports the number of file fragments for the main database file, as reported by the operating system. Excessive disk fragmentation of the disk volume that holds the database can cause performance problems because fragmentation can lead to additional disk latency during I/O operations.

The problem of file system fragmentation is independent of table page fragmentation within the database. The analysis and resolution of these separate problems are performed using different methods and tools. The number of file fragments is reported only for Windows-based SQL Anywhere database servers. Unix file systems, including Linux file systems, incur significantly fewer fragmentation issues than Windows-based systems.

On Windows platforms, database files generally consist of more than one fragment. When you receive this message for a database, the number of fragments relative to the size of the database file helps you determine whether the system administrator needs to take any action. For example, a 100 GB database file that is composed of 25 fragments should not be considered a serious problem, but a 50 MB database file consisting of 300 disk fragments may be affected by performance problems. To eliminate file fragmentation problems, put the database on a disk partition by itself and then periodically run one of the available Windows disk defragmentation utilities.

See also

- "Reduce fragmentation" [*SQL Anywhere Server - SQL Usage*]
- "Reduce table fragmentation" [*SQL Anywhere Server - SQL Usage*]
- "Reduce table fragmentation" [*SQL Anywhere Server - SQL Usage*]
- "DBFileFragments database property" on page 663

Performance warning: In database %3, table %1 autoincrement column %2 is not indexed"

This warning is reported when a permanent base table contains an autoincrement column and there is no index on the column. The warning is generated only in cases where the maximum identity value for that column, as stored in the MAX_IDENTITY column in the ISYSTABCOL catalog table, is unknown or if it becomes invalid. This situation can occur if an existing column has been altered to be an autoincrement column or if the global autoincrement settings for the database have changed.

See also

- "ISYSTABCOL system table" [*SQL Anywhere Server - SQL Reference*]
- "The AUTOINCREMENT default" [*SQL Anywhere Server - SQL Usage*]
- "The GLOBAL AUTOINCREMENT default" [*SQL Anywhere Server - SQL Usage*]

Performance warning: View %1 in database %2 was invalidated due to DDL operations on one of its referenced objects

This warning is reported when a DDL operation invalidates an existing view. If a view is dependent on a schema object that has been modified, and the view subsequently fails to recompile, the status of that view remains INVALID. With each subsequent reference from within a query, the database server attempts to recompile the view. See "Dependencies and schema-altering changes" [*SQL Anywhere Server - SQL Usage*].

See also

- "Working with views" [*SQL Anywhere Server - SQL Usage*]
- "Regular view statuses" [*SQL Anywhere Server - SQL Usage*]

Troubleshooting network communications

Network software involves several different components, increasing the likelihood of problems. Although some tips concerning network troubleshooting are provided here, the primary source of assistance in network troubleshooting should be the documentation and technical support for your network communications software, as provided by your network communications software vendor.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Use logging

Specifying the -z database server option displays diagnostic communication messages, and other messages to the database server messages window for troubleshooting purposes. These messages can help you diagnose how a connection is failing, what connection parameters are used for the connection attempt, and what communication links are being used. See "-z dbeng12/dbsrv12 server option" on page 243.

Ensure that you are using compatible protocols

Ensure that the client and the database server are using the same protocol. The -x option for the server selects a list of protocols for the server to use, and the CommLinks (LINKS) connection parameter does the same for the client application.

You can use these options to ensure that each application is using the same protocol.

By default, the network database server uses all available protocols. The server supports client requests on any active protocol. By default, the client only uses the shared memory protocol. The client can use all available protocols by setting the CommLinks (LINKS) connection parameter to all.

See also

- "-x dbeng12/dbsrv12 server option" on page 236
- "Host connection parameter" on page 291
- "CommLinks (LINKS) connection parameter" on page 272

Ensure that you have current drivers

Old network adapter drivers can be a source of communication problems. You should ensure that you have the latest version of your network adapter. You should be able to obtain current network adapter drivers from the manufacturer or supplier of the card.

Testing the TCP/IP protocol

The ping utility can be useful to test that TCP/IP is installed and configured properly.

Using ping to test the IP layer

Each IP layer has an associated address—for IPv4, this address is a four-integer, dot-separated number (such as 191.72.109.12). Ping takes an IP address as an argument and attempts to send a single packet to the address.

First, determine if your own computer is configured correctly by using the ping utility to detect your computer. If your IP address is 191.72.109.12, you would run the following command and wait to see if the packets are routed at all:

```
ping 191.72.109.12
```

If the packets are routed, output similar to the following appears:

```
Pinging 191.72.109.12 with 32 bytes of data:
Reply from 191.72.109.12: bytes=32 time<.10ms TTL=32
Reply from 191.72.109.12: bytes=32 time<.10ms TTL=32
Reply from 191.72.109.12: bytes=32 time<.10ms TTL=32
...
```

This means that the computer is able to route packets to itself. This is reasonable assurance that the IP layer is set up correctly. You could also ask someone else running TCP/IP for their IP address and try using the ping utility to detect their computer.

You should ensure that you can ping the computer running the database server from the client computer before proceeding.

Testing the IP layer on IPv6

If you are attempting to connect to a host on an IPv6 network, you must first ensure that IPv6 is installed on the client computer. On Windows XP, run the command `ipv6 install` to install IPv6. IPv6 is installed by default on Windows Vista. Installing IPv6 is different on each Unix operating system; consult the operating system documentation for instructions on enabling IPv6.

Once IPv6 is installed and enabled, you can use the `ping6` command to do the same thing as the ping command described above. For example:

```
ping6 fe80::213:ceff:fe24:ca6

Pinging fe80::213:ceff:fe24:ca6
from fe80::213:ceff:fe24:ca6%6 with 32 bytes of data:

Reply from fe80::213:ceff:fe24:ca6%6: bytes=32 time<1ms
Reply from fe80::213:ceff:fe24:ca6%6: bytes=32 time<1ms
Reply from fe80::213:ceff:fe24:ca6%6: bytes=32 time<1ms
...
```

Diagnosing wiring problems

Faulty network wiring or connectors can cause problems that are difficult to track down. Try recreating problems on a similar computer with the same configuration. If a problem occurs on only one computer, it may be a wiring problem or a hardware problem.

A checklist of common problems

The following list presents some common problems and their solutions.

For information about troubleshooting connections to the database or database server, see "Troubleshooting connections" on page 138 and "Troubleshooting database server startup" on page 998.

If you receive the message `Database server not found` when trying to connect, the client cannot find the database server on the network. Check for the following problems:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- Under the TCP/IP protocol, clients search for database servers by broadcasting a request. Such broadcasts will typically not pass through gateways, so any database server on a computer in another (sub)network, will not be found. If this is the case, you must supply the host name of the computer on which the server is running using the HOST (IP) protocol option.
- A firewall between the client and server may be preventing the connection. See "Connecting across a firewall" on page 80.
- The personal server only accepts connections from the same computer. If the client and server are on different computers, you must use the network server.
- Your network drivers are not installed properly or the network wiring is not installed properly.
- If you receive the message `Unable to initialize requested communication links`, one ore more links failed to start. The probable cause is that your network drivers have not been installed. Check your network documentation to find out how to install the driver you want to use.
- The server should use the TCP/IP protocol if you are connecting via jConnect.
- If you are trying to connect to a database server on your local computer, choose **Connect To A Running Database On This Computer** in the **Connect** window. You can choose **Connect To A Running Database On Another Computer** if you are trying to connect to a database server running on a computer other than your local computer.

For more information about network protocol options, see "Network protocol options" on page 311.

Adjusting timeout values

If you are experiencing problems with connections unexpectedly disconnecting, consider adjusting either the liveness or the idle timeout values.

See also

- "LivenessTimeout (LTO) connection parameter" on page 297
- "-tl dbeng12/dbsrv12 server option" on page 226
- "Idle connection parameter" on page 292
- "-ti dbeng12/dbsrv12 server option" on page 226

Monitoring your database

This section describes how to use the SQL Anywhere Monitor to monitor your SQL Anywhere databases and MobiLink servers. It also describes how to set up and configure the SQL Anywhere SNMP Extension Agent.

SQL Anywhere Monitor

The SQL Anywhere Monitor, also referred to as the Monitor, is a browser-based administration tool that provides you with information about the health and availability of SQL Anywhere databases, MobiLink servers, MobiLink server farms, and Relay Server farms.

This section describes how to use the Monitor to collect metrics about *SQL Anywhere* databases. For information about using the Monitor with:

- MobiLink servers and MobiLink server farms, see "SQL Anywhere Monitor for MobiLink" [*MobiLink - Server Administration*].
- Relay Server farms, see "SQL Anywhere Monitor for Relay Server" [*Relay Server*].

The Monitor provides the following functionality:

- **Constant data collection** Unlike many of the other administration tools available with SQL Anywhere, the Monitor collects metrics all the time, even when you are not logged in to the browser. The Monitor collects metrics until you shut it down.
- **Email alert notification** As the metrics are collected, the Monitor examines the metrics and can send email alerts when it detects conditions that indicate something is wrong with a resource.
- **Browser-based interface** At any time, you can log in to the Monitor using a browser to review alerts and metrics that have been collected.
- **Monitor multiple databases, MobiLink servers, MobiLink server farms, and Relay Server farms** From one tool, you can simultaneously monitor multiple resources running on the same or different computers.
- **Minimal performance impact** The Monitor can be used routinely in development and production environments because monitoring does not degrade performance.

The Monitor is designed to help any type of user, whether they are a DBA or not, who is responsible for such tasks as:

- Ensuring that a database is connected to the network.
- Ensuring that there is enough disk space or memory available for a database.
- Ensuring that users aren't blocked or that queries aren't taking too long to run.

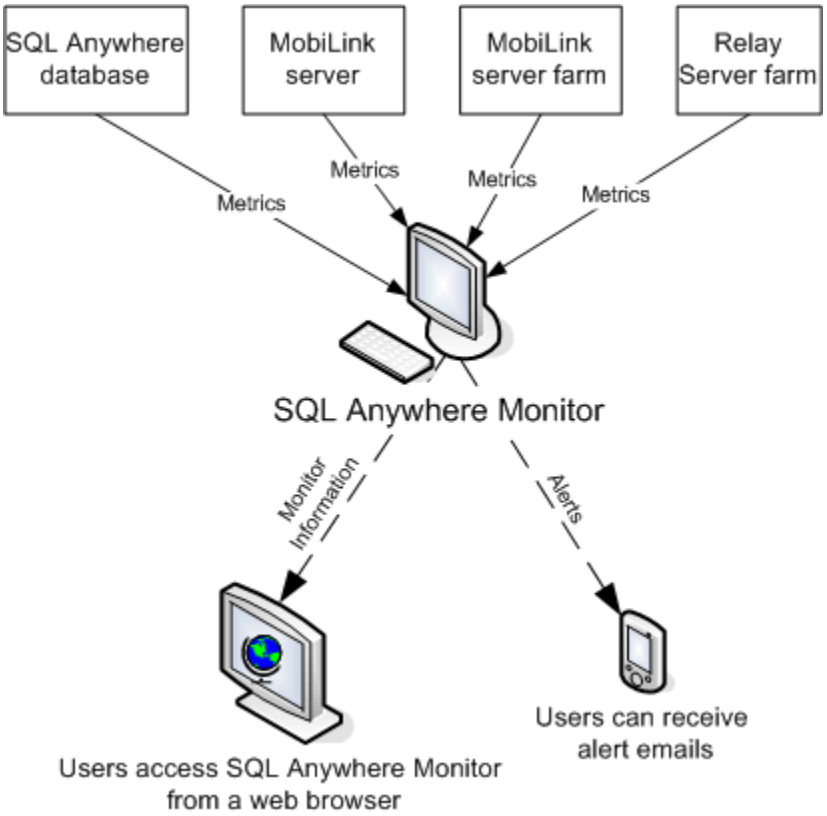
See also

For information about other administration and performance tools that are available for SQL Anywhere databases, see:

- "Application profiling" [*SQL Anywhere Server - SQL Usage*]
- "SQL Anywhere Console utility (dbconsole)" on page 848
- "Monitoring statistics using Sybase Central Performance Monitor" [*SQL Anywhere Server - SQL Usage*]
- "Monitor statistics using Windows Performance Monitor" [*SQL Anywhere Server - SQL Usage*]
- "Performance Statistics utility (dbstats) (Unix)" on page 824

Monitor architecture

The Monitor collects metrics and performance data from SQL Anywhere databases, MobiLink servers, MobiLink server farms, and Relay Server farms running on other computers, while a separate computer accesses the Monitor via a browser.



The Monitor is a Flash-based application that is served to a web browser via the SQL Anywhere built-in HTTP server. You interact with the Monitor through its browser interface.

There are two editions of the Monitor:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **SQL Anywhere Monitor Developer Edition** The Monitor Developer Edition is intended for development and testing use. It is installed by default with SQL Anywhere and it uses the installed SQL Anywhere on the back-end.

- **SQL Anywhere Monitor Production Edition** This Production Edition is intended for deployment and production use. It is installed separately, runs as a service, and includes a fully-contained SQL Anywhere installation. This edition is only available for certain editions of SQL Anywhere. See "Installing the SQL Anywhere Monitor in a production environment" on page 1061.

Requirements

For the computer where the Monitor is installed

- To run the SQL Anywhere Monitor on a Windows system that has the Windows Firewall enabled, you must add a port exception for port 4950.

- The SQL Anywhere Monitor reserves 384 MB of virtual address space when you start the Monitor. When you start the Monitor on Linux, you must ensure that you have least this amount available.

- To monitor resources that are secured using ECC encryption or FIPS-certified encryption, you need a separate license. See "Securing the Monitor" on page 1063.

- The Monitor can run on the same computer as the resources it is monitoring, but it is recommended, particularly in production environments, that you run the Monitor on a different computer to minimize the impact on the database server or other applications.

- It is recommended in production environments that you run the Monitor Production Edition. See "Installing the SQL Anywhere Monitor in a production environment" on page 1061.

- When running the Monitor Developer Edition, you must have SQL Anywhere 12.0.0 installed. The Monitor Developer Edition uses the installed SQL Anywhere on the back-end.

For the computer accessing the Monitor

- Install the latest version of Adobe Flash Player that is available for your operating system. The Monitor is backwards compatible with version 10 of Adobe Flash Player. To determine the correct version, see http://www.adobe.com/products/flashplayer/systemreqs/.

- Enable JavaScript in your browser.

- Ensure that the computer where you are using a browser to access the Monitor is connected to the network where the Monitor is installed.

  You can access the Monitor from the same computer as it is running on, but it is recommended, particularly in production environments, that you access the Monitor from a different computer.

Limitations

- You can use the Monitor to collect metrics about the following versions of SQL Anywhere databases, MobiLink servers, MobiLink server farms, and Relay Server farms:

  - SQL Anywhere 9.0.2, 10.0.0, 10.0.1, 11.0.0, 11.0.1, and 12.0.0
  - MobiLink 11.0.0 with at least the first EBF applied, 11.0.1, and 12.0.0
  - Relay Server 12.0.0
- You can only run one edition of the Monitor on a computer at a time.
- On Linux, the Monitor Developer Edition can only be run by the user who installed it.
- On Linux, only the user with administrator privilege can install and run the Monitor Production Edition.
- You cannot use the Monitor to optimize queries or determine the speed of your application. If you are interested in tuning database and application performance, you can use such tools as the **Application Profiling Wizard**, the **Sybase Central Performance Monitor**, or the **Windows Performance Monitor**.

See also

- "Monitor quick start" on page 1010

Monitor quick start

The following steps are required to set up SQL Anywhere database monitoring:

1. Start the database that you want to monitor (if it is not already running).
2. Install the Monitor on a computer that is always connected to your network.

   The Monitor can run on the same computer as the resources it is monitoring, but it is recommended, particularly in production environments, that you run the Monitor on a different computer to minimize the impact on the database server, or other applications.
3. Start the Monitor and open it in your browser. See "Start the Monitor" on page 1021.
4. Log in to the Monitor as an administrator. The default user is an administrator user with the name **admin** and the password **admin**.

   Note
   You must be logged in to the Monitor as an administrator to perform the following tasks. See "Monitor users" on page 1048.
5. As an administrator, in the left navigation menu, choose **Tools** » **Administration** and add a SQL Anywhere database as a resource to be monitored. See "Add resources" on page 1033.
6. As an administrator, add new users and change the password for the admin user. See "Create Monitor users" on page 1049.
7. As an administrator, configure alert thresholds for the resources. See "Alerts" on page 1052.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

8. As an administrator, configure the backup and maintenance schedule. See "Back up the Monitor" on page 1058.

9. Click **Close** to exit **Administration**.

10. Click **Overview**. In the **Resource List** widget, you will see the resources that are being monitored. See "Overview dashboard" on page 1026.

Tutorial: Using the Monitor

Use this tutorial to set up monitoring of a SQL Anywhere database. This tutorial uses the Monitor Developer Edition.

Lesson 1: Log in to the Monitor as the default administrator

To start and log in to the Monitor

1. Start the Monitor. The following steps assume that the Monitor is not currently running in the background.

   To start the Monitor Developer Edition (Windows): Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **SQL Anywhere Monitor**.

   To start the Monitor Developer Edition (Linux): Run the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory. Run the following:

   ```
   samonitor.sh launch
   ```

   The Monitor starts collecting metrics and a browser opens the default URL where you can log in to the Monitor: *http://localhost:4950*.

   Note
   If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running.

2. Log in to the Monitor as the default *administrator* user.

   In the **User Name** field, type **admin**, and in the **Password** field, type **admin**.

Note
You must be logged in to the Monitor as an administrator to perform the following lessons in the tutorial. Read-only and operator users do not have permission to perform all the tasks.

To check your Monitor user type

1. Log in to the Monitor.
2. Choose **Tools** » **User Settings** and review the **User Type** setting.

See "Monitor users" on page 1048.

By default, the Monitor opens the **Overview** dashboard that contains two widgets:

1. The **Resource List** widget lists the resources that are being monitored. When you first open the Monitor, it is only monitoring itself via the default resource, named **SQL Anywhere Monitor**. You cannot modify this resource, nor can you stop monitoring it.
2. The **Alerts List** widget lists any alerts from the monitored resources.

See also

- "Lesson 2: Set up the Monitor to monitor a database" on page 1012
- "Start the Monitor" on page 1021

Lesson 2: Set up the Monitor to monitor a database

In this lesson, you start the SQL Anywhere sample database, *demo.db*, and then add this database as a resource to be monitored.

To add a database resource to the Monitor

1. Start the SQL Anywhere sample database. From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **SQL Anywhere** » **Network Server Sample**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

2. Log in to the Monitor as an administrator. See "Lesson 1: Log in to the Monitor as the default administrator" on page 1011.

3. Choose **Tools** » **Administration**. which is located on the left sidebar.

4. Click **Resources**, and then click **Add**.

5. Select **SQL Anywhere Server**, and then click **Next**.

6. In the **Name** field, type **demo12**, and then click **Next**.

7. In the **Host** field, type **localhost**, and in the **Server** field, type **demo12**.

8. Click **Create**.

9. When you are prompted for the required authorization, in the **DBA User ID** field, type **DBA**, and in the **Password** field, type **sql**. Click **OK**.

   When you add a database as a resource to be monitored, you must supply the DBA user ID and password for the database. These credentials are used to:

   - Connect to the database.
   - Create a new user named **sa_monitor_user**. The Monitor uses the sa_monitor_user to connect to the database and monitor it.
   - Install the database objects needed by the **sa_monitor_user** to monitor the database. For a list of the objects installed, see "Installed objects" on page 1060.
   - Discard from the Monitor the DBA credentials. Because the **sa_monitor_user** is added to the database being monitored, it is not necessary to store the DBA credentials anywhere outside the database that is being monitored. See "Installed objects" on page 1060.

   The resource is added and monitoring of the resource starts automatically.

10. Click **OK**.

11. Click **Close**.

12. Choose **Dashboards** » **Overview**.

    The **demo12** resource appears in the **Resource List**.

13. Click **demo12** to open the demo12 dashboard and view the collected metrics.

See also

- To collect metrics from a MobiLink server or MobiLink server farm, see "Lesson 2: Set up the Monitor to monitor a MobiLink server" [*MobiLink - Server Administration*].
- To collect metrics from a Relay Server farm, see "Lesson 2: Set up the Monitor to monitor a Relay Server farm" [*Relay Server*].

Lesson 3: Test an alert

In this lesson, you intentionally trigger an alert so you can practice handling alerts.

To view and resolve an alert

1. Trigger an alert by shutting down the resource.

   For example:

   a. On Windows, double-click the network server icon in the system tray for the **demo12** database server.

   b. Click **Shut Down** in the database server messages window.

   c. Click **Yes**.

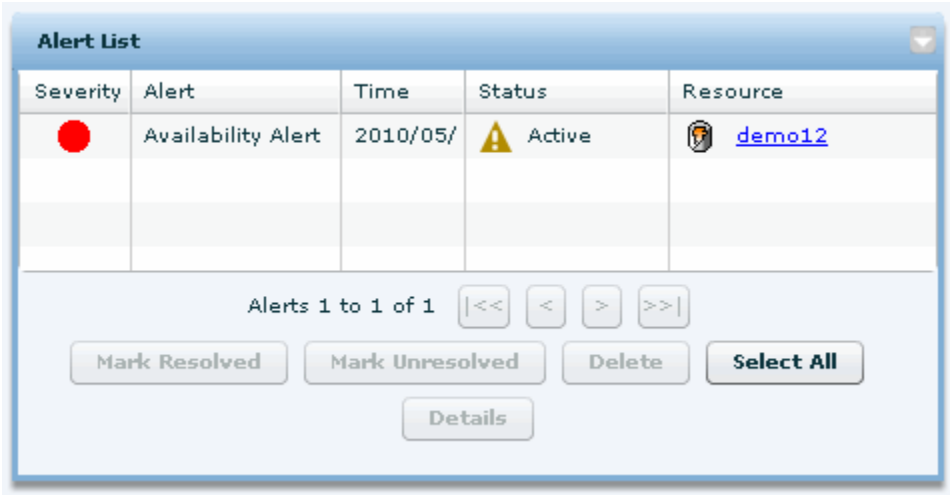
2. Log in to the Monitor as an administrator. See "Lesson 1: Log in to the Monitor as the default administrator" on page 1011.

3. Choose **Dashboards** » **Overview**.

   In the **Resource List**, the **Status** for the **demo12** resource changes to a red circle with a white x through it. This icon indicates that the resource is unavailable.

   In the **Alerts** widget, an **Availability Alert** appears and its status is **Active**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

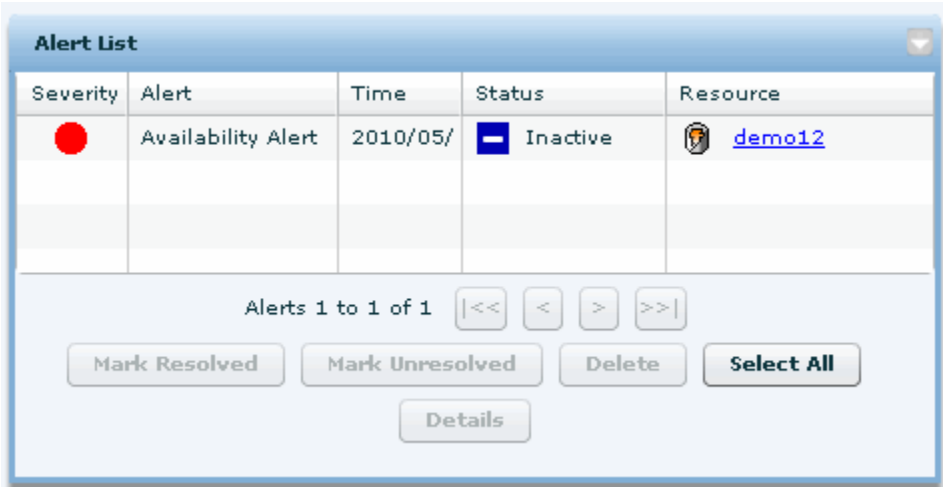


It can take a few seconds for these changes in state and status to occur. By default, the Monitor collects information from the resource every 30 seconds. See "Collection intervals" on page 1042.

4. In the **Alerts** dashboard, click the **Availability Alert**, and then click **Details** to read the description.
5. Click **OK** to close the alert.
6. Restart the sample database.

   Start the SQL Anywhere sample database. From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **SQL Anywhere** » **Network Server Sample**.

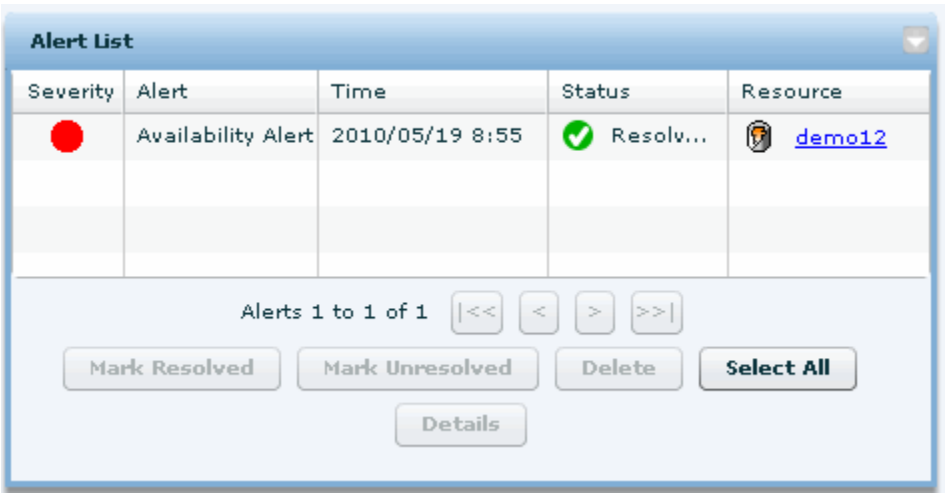
   In the Monitor, in the **Resource List**, the **Status** for the **demo12** resource changes to a yellow triangle. This icon indicates that the resource is being monitored and it has an alert. In the **Alerts List**, the **State** of the **Availability Alert** changes to **Inactive**. An inactive alert indicates that the issue that triggered the alert is no longer present, but the alert has not been resolved or deleted.



7. Resolve the alert by selecting the alert and clicking **Mark Resolved**.

> **Note**
> Only administrators and operators can resolve and delete alerts. See "Monitor users" on page 1048.

Now in the **Resource List**, the **Status** for the **demo12** resource is blank. No icon in the **Status** column indicates that the resource is being monitored and it has no alerts. In the **Alerts List**, the **State** of the **Availability Alert** changes to **Resolved by *admin*** where *admin* is your Monitor user name.



8. Delete the alert by selecting the alert and clicking **Delete**.

When prompted, click **Yes** to confirm the deletion.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Lesson 4: Set up the Monitor to send emails when alerts occur

When an alert occurs, it is always listed in the **Alerts List** widget on the dashboard for the particular resource. See "Lesson 3: Test an alert" on page 1014.

In this lesson, you set up the Monitor to send you an email whenever an alert occurs.

To set up email notification

1. Log in to the Monitor as an administrator. See "Lesson 1: Log in to the Monitor as the default administrator" on page 1011.

2. Add a user that can receive emails.
   a. Choose **Tools** » **Administration**.
   b. Choose **Users**, and then click Add.
   c. In the User Name field, type **JoeSmith**.
   d. In the **Password** and the **Confirm Password** fields, type **password**.
   e. In the **Email** field, enter a valid email address.
   f. For the **User Type**, select **Operator**.

      An operator can receive alerts via email and can resolve and delete alerts. This user can access most of the Monitor widgets but it cannot not access the **Administration** window.
   g. Click **Next**.
   h. When prompted to choose the resources you are interested in, click **Check All**.
   i. Click **Save**.

      The new user is created and you are returned to the **Administration** window.

3. Configure email alert notification.

   Note
   You must be logged in to the Monitor as an administrator to perform the following tasks. Only administrators can configure the Monitor to send emails. See "Monitor users" on page 1048.

   a. In **Administration** window, select **Configuration** and click **Edit**.
   b. On the **Alert Notifications** tab, select **Send Alert Notifications By Email**.
   c. Configure other settings as required. See "Enable the Monitor to send alert emails" on page 1057.
   d. Test that you have properly configured email notification.

      Click **Send Test Email**.
   e. When prompted, enter an email address to send the test email to and click **OK**.

      A test email is sent to the email address specified.
   f. Click **Save**.
   g. Click **Close**.

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

When an alert occurs, an email is sent to the specified user with information about the alert.

Lesson 5: Add a new dashboard and widgets

Dashboards are specific to each user. Any user can add, edit or delete their own dashboards. In addition, any user can add, edit, or delete the widgets that exist in their dashboards. By default, when a dashboard is created, it is populated with default widgets. In this lesson, you add a dashboard to the Monitor, and then add widgets.

To add a new dashboard

1. Log in to the Monitor.
2. Choose **Dashboards** » **Add new**.
3. Select **A Dashboard To Monitor The Following Resource**, and choose the resource.
4. In the **Dashboard Name** field, type **user_joe**.
5. In the **Number Of Columns** field, type **2**.
6. Click **OK**.

   A new dashboard appears, with the 4 following widgets: **Alert List**, **Key Performance Metrics**, **Server Info**, and **Connections**.

The following procedure describes how to create a new metrics widget that displays graphs, instead of the default spark lines.

To add a metrics display widget with graphs

1. In the upper right corner of the **user_joe** dashboard, click **Customize**, and then click **Add Widget**.
2. Select **Metrics**, and then click **Next**.
3. In the **What Do You Want To Name This Widget?** field, type **Metrics display**.
4. In the **Which Resource Are You Interested In?** field, choose **demo12**.
5. In the **What Kind Of Display Do You Want To See?** field, choose **Graph**.
6. For **Which metrics do you want to see?**, select **CPU Usage**, **Memory Metrics** » **Cache Size**, and **Queries Processed**.
7. Click **Create**.

   A widget called **Metrics display** appears in the dashboard.

   To maximize the size of a widget, in the widget pane, click the dropdown menu arrow, and choose **Maximize**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To view details on the graph, position your cursor above specific points on the graph.

See also

- "Dashboards" on page 1028
- "Widgets" on page 1029

Lesson 6: Closing database connections

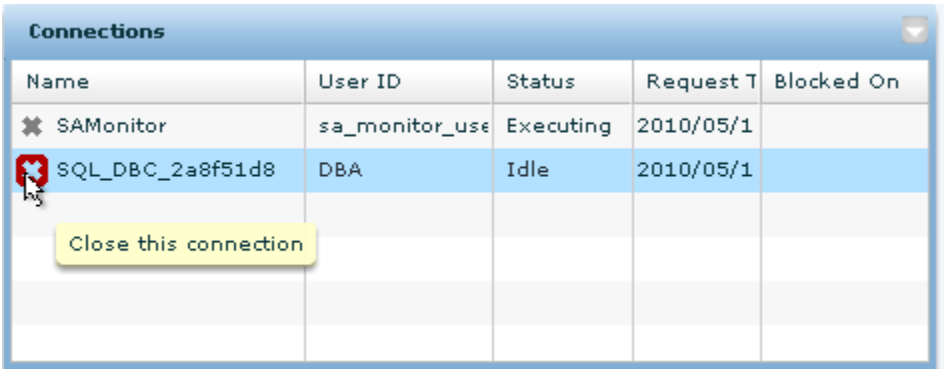
In this lesson, you use the Monitor to close a connection to a resource database. To close a connection, you need the resource database's DBA user ID and password.

To close database connections

1. Log in to the Monitor as an administrator. See "Lesson 1: Log in to the Monitor as the default administrator" on page 1011.

2. Open the **demo12** dashboard.

3. In the **Connections** widget, click the **x** beside the name of the connection you want to close.



4. When you are prompted for the required authorization, in the **DBA User ID** field, type **DBA**, and in the **Password** field, type **sql**.

   The Monitor uses the DBA user ID and password to connect to the resource database to drop the connection. The DBA user ID and password are not kept by the Monitor.

5. Click **OK**.

   The connection is removed from the **Connections** widget.

Lesson 7: Cleanup

In this lesson, you remove the demo12 resource, which deletes the collected metrics and stops data collection. In a production environment when you want to continue monitoring your database, you leave both the database and the Monitor running.

Note
You must be logged in to the Monitor as an administrator to perform the following tasks. Only administrators can remove resources. You cannot delete the **SQL Anywhere Monitor** resource. See "Monitor users" on page 1048.

To stop monitoring a resource

1. Log in to the Monitor as an administrator.
2. Remove the **demo12** resource.
   a. Choose **Tools** » **Administration**.
   b. Select **Resources**.
   c. Select the **demo12** resource, and click **Stop**.
   d. Click **Remove**.
   e. Click **Yes** to confirm that you want to remove the resource.

   Note
   When you remove a database resource, the Monitor does **not** delete the monitoring objects installed in the database. To delete these objects, you must connect as a user with DBA authority to the database, and then execute the following command:

   ```
   DROP USER sa_monitor_user;
   ```

   See "Deleting monitoring objects" on page 1061.

   f. Click **Close**.
3. Log out of the Monitor.

   Click **Logout**.
4. Close the browser window where you are viewing the Monitor.
5. Exit the Monitor and stop monitoring.
   - On Windows, in the system tray, right-click the SQL Anywhere Monitor icon and choose **Exit SQL Anywhere Monitor**.
   - On Linux, run the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory:

     ```
     samonitor.sh stop
     ```

   The Monitor stops collecting metrics for all resources.
6. Shut down the SQL Anywhere database.
   a. Double-click the network server icon in the system tray for the demo12 database server.
   b. Click **Shut Down** in the database server messages window.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

c. Click **Yes**.

Start the Monitor

When you start the Monitor, it connects to the resources and collects metrics for *all* resources in the Monitor.

The Monitor is designed to run silently in the background. You interact with the Monitor through its browser interface. It is recommended that you leave the Monitor running continuously in the background to collect the metrics.

Note
You can only run one edition of the Monitor on a computer at a time.

To start the Monitor Developer Edition (Windows)

1. On the computer where the Monitor is installed, choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **SQL Anywhere Monitor**.

   The Monitor connects to the resources and begins collecting metrics for all resources in the Monitor, and:

   - A browser opens the default URL for logging in to the Monitor: *http://localhost:4950*.

     Note
     If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running.

     See "Log in remotely to the Monitor" on page 1025.

   - The SQL Anywhere Monitor icon appears in the system tray.

2. Log in.

   In the browser, enter your user name and password for the Monitor. The user name and password for the Monitor are case sensitive. There are three different types of users: administrators, operators, and read-only users. Each type of user has different permissions. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 1048.

   You can select **Remember Me On This Computer** to have your session persist for two weeks or until you log out. If **Remember Me On This Computer** is cleared, your session expires when you close the browser or log out.

To start the Monitor Developer Edition (Linux)

1. On the computer where the Monitor is installed, start the Monitor.

   Run the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory:

   ```
   samonitor.sh launch
   ```

The Monitor connects to the resources and begins collecting metrics for all resources in the Monitor, and a browser opens the default URL for logging in to the Monitor: *http://localhost:4950*.

Note
If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 1025.

2. Log in.

   In the browser, enter your user name and password for the Monitor. The user name and password for the Monitor are case sensitive. There are three different types of users: administrators, operators, and read-only users. Each type of user has different permissions. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 1048.

   You can select **Remember Me On This Computer** to have your session persist for two weeks or until you log out. If **Remember Me On This Computer** is cleared, your session expires when you close the browser or log out.

To start the Monitor Production Edition (Windows)

1. On Windows, by default, the Monitor Production Edition runs automatically as a service when installed. Because it runs as a service, this Monitor also starts automatically when the computer starts. If you stop the Monitor, you can restart the Monitor service by doing one of the following steps:
   - On the computer where the Monitor is installed, choose **Start** » **Programs** » **SQL Anywhere 12 Monitor** » **SQL Anywhere Monitor**.
   - Run the *samonitor* batch file from the *bin32* or *bin64* directory in the Monitor installation directory. This batch file starts the Monitor service:

     ```
     samonitor.bat start
     ```

   The Monitor starts collecting metrics.

2. Browse to the URL for logging in to the Monitor. The default URL is *http://localhost:4950*.

   Note
   If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 1025.

3. Log in.

   When prompted, enter your user name and password for the Monitor. The user name and password for the Monitor are case sensitive. There are three different types of users: administrators, operators, and read-only users. Each type of user has different permissions. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 1048.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can select **Remember Me On This Computer** to have your session persist for two weeks or until you log out. If **Remember Me On This Computer** is cleared, your session expires when you close the browser or log out.

To start the Monitor Production Edition (Linux)

1. On Linux, by default, the Monitor Production Edition runs automatically as a service when installed. Because it runs as a service, this Monitor also starts automatically when the computer starts. If you stop the Monitor, you can restart the Monitor service by running the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory. This script starts the Monitor service:

   ```
   samonitor.sh launch
   ```

   The Monitor starts collecting metrics and the browser opens.

2. Browse to the URL for logging in to the Monitor. The default URL is *http://localhost:4950*.

   Note
   If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 1025.

   The Monitor connects to the resources and begins collecting metrics for all resources in the Monitor, and a browser opens the default URL for logging in to the Monitor: *http://localhost:4950*.

   Note
   If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 1025.

3. Log in.

   When prompted, enter your user name and password for the Monitor. The user name and password for the Monitor are case sensitive. There are three different types of users: administrators, operators, and read-only users. Each type of user has different permissions. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 1048.

   You can select **Remember Me On This Computer** to have your session persist for two weeks or until you log out. If **Remember Me On This Computer** is cleared, your session expires when you close the browser or log out.

See also

- "Stop the Monitor" on page 1024
- "To stop the Monitor Production Edition" on page 1024
- "Log out from the Monitor" on page 1025
- "Overview dashboard" on page 1026

Stop the Monitor

Stopping the Monitor stops the collection of metrics for all resources.

Caution
In most cases, it is recommended that you leave the Monitor running, but close the browser. Closing the browser does not stop the collection of metrics.

To stop monitoring a specific database, see "Stop monitoring resources" on page 1038.

To stop the Monitor Developer Edition (Windows)

- In the system tray, right-click the SQL Anywhere Monitor icon and choose **Exit SQL Anywhere Monitor**.

  The Monitor stops collecting metrics for all resources.

  To restart the Monitor, see "Start the Monitor" on page 1021.

To stop the Monitor Developer Edition (Linux)

- Run the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory:

  ```
  samonitor.sh stop
  ```

  The Monitor stops collecting metrics for all resources.

  To restart the Monitor, see "Start the Monitor" on page 1021.

To stop the Monitor Production Edition

By default, the Monitor Production Edition runs automatically as a service when installed on Windows and Linux. To stop the Monitor and the collection of all resource metrics, you must stop the service.

To stop the Monitor service (Windows)

- To stop the Monitor service, run *samonitor.bat* from the *bin32* directory in the Monitor installation directory:

  ```
  samonitor.bat stop
  ```

  This batch file stops the Monitor service. The Monitor stops collecting metrics for all resources.

  To restart the Monitor service, see "Start the Monitor" on page 1021.

To stop the Monitor service (Linux)

- To stop the Monitor service, run the *samonitor.sh* script from the *bin32* or *bin64* directory in the Monitor installation directory:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
samonitor.sh stop
```

This script stops the Monitor service. The Monitor stops collecting metrics for all resources.

To restart the Monitor service, see "Start the Monitor" on page 1021.

See also

- "Start the Monitor" on page 1021
- "Log in remotely to the Monitor" on page 1025
- "Log out from the Monitor" on page 1025
- "Overview dashboard" on page 1026

Log in remotely to the Monitor

The computer that you are using to log in to the Monitor must be connected to the network where the Monitor is running.

To log in to the Monitor

1. On the computer where the Monitor is installed, start the Monitor. See "Start the Monitor" on page 1021.

2. On the computer that is accessing the Monitor, browse to the default URL for logging in to the Monitor: **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. For example, *http://samplehost:4950*.

3. When prompted, enter your user name and password for the Monitor. The user name and password for the Monitor are case sensitive. See "Monitor users" on page 1048.

See also

- "Start the Monitor" on page 1021
- "Stop the Monitor" on page 1024
- "Log out from the Monitor" on page 1025
- "Overview dashboard" on page 1026

Log out from the Monitor

Logging out from the Monitor has *no* effect on the collection of metrics. If you want to stop collecting metrics, then:

- Stop monitoring the resource. See "Stop monitoring resources" on page 1038.

- Exit the Monitor. See "Stop the Monitor" on page 1024.

To log out from the Monitor

- Click **Logout** in the upper right corner.

  If you select **Remember Me On This Computer** when you log in to the Monitor, then closing the browser does not log you out of the Monitor.

See also

- "Start the Monitor" on page 1021
- "Stop the Monitor" on page 1024
- "Log in remotely to the Monitor" on page 1025
- "Overview dashboard" on page 1026

Overview dashboard

The **Overview** dashboard provides an overview of the health and availability of the resources (for example, SQL Anywhere databases) being monitored.

By default, the **Overview** dashboard contains the **Resource List** widget and the **Alert List** widget.

1: Resource List widget

The **Resource List** widget contains a table that lists the resources being monitored, as well as an indication about the overall health of each resource. The table always contains the default resource, named **SQL Anywhere Monitor**, which reports on the health of the Monitor itself. You cannot modify the **SQL Anywhere Monitor** resource, nor can you stop monitoring it.

To view detailed information about a resource

- Click the resource name in the **Resource List** widget.

  The dashboard for the selected resource opens. See "Dashboards" on page 1028.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

In the **Resource List** widget, the **Status** column provides information about the connections between the Monitor and its resources. The **Status** column indicates whether the resource requires someone to perform an action on it.

A resource has one of the following statuses:

Icon	Status	Description
No icon present	**Healthy**	There are no unresolved alerts for the resources.
	**Alerts Present**	There are one or more unresolved alerts for the resource.
	**Unavailable**	The resource is unavailable. For example, the resource is down.
	**Monitoring Stopped**	The resource is not being monitored because of a blackout or because a user manually stopped monitoring the resource.

2: Alert List widget

The **Alert List** widget contains the alerts for the monitored databases.

To view detailed information about an alert

- Select the alert in the **Alert List**, and then click **Details**.

  A window opens showing the details of the alert.

An alert has one of the following statuses:

Icon	Status	Description
	Active	Active alerts are alerts where the alert condition still applies. No one has resolved the alert.
	Inactive	An inactive alert indicates that the issue that triggered the alert is no longer present, but the alert has not been resolved or deleted.
	**Resolved**	An administrator or operator has marked the alert resolved.

An alert has one of the following levels of severity:

Icon	Severity	Description
	High severity	High severity alerts indicate problems that require a user's immediate attention. For example, when a resource exceeds the low disk space threshold, a high severity alert is issued.
	Medium severity	Medium severity alerts indicate problems that require a user's attention as the problems could escalate. For example, when a resource exceeds the CPU usage threshold, a medium severity alert is issued.
	Low severity	Low severity alerts indicate problems. For example, when a resource has a failed connection, a low severity alert is issued.

See "Alerts" on page 1052.

Dashboards

Dashboards are specific to each user. Any user can add, edit, or delete their dashboards.

To add a new dashboard

1. Log in to the Monitor.
2. In the **Dashboards** pane, click **Add New**.
3. Follow the instructions in the window to add a dashboard.
4. Click **OK**.

   The Monitor creates and opens the new dashboard. The name of the dashboard also appears under the **Dashboards** list on the left navigation menu.

To edit or delete a dashboard

1. Open the dashboard.
2. In the upper right corner of the dashboard, click **Customize**, and then choose either:
   - **Settings** Edits the dashboard's settings.
   - **Delete** Deletes the dashboard.

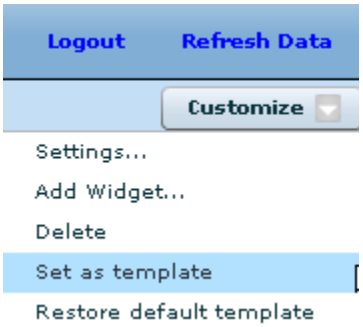
Dashboard templates

By default, when a dashboard is created, it is populated with default widgets. You can change the default widgets that appear with a new dashboard by configuring the Monitor to use a specified dashboard's widgets and layout. See "Widgets" on page 1029.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To create a dashboard template

1. Open the dashboard that you want to use as the template.
2. In the upper right corner of the dashboard, click **Customize**, and then click **Set As Template**.



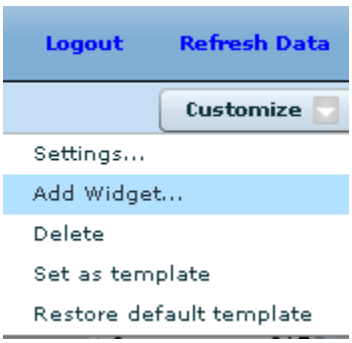
Now when you add a new dashboard, it contains the same widgets and layout as the template dashboard.

Widgets

Any user can add, edit, or delete the widgets that exist in their dashboards. By default, when a dashboard is created, it is populated with default widgets.

To add widgets

1. Open the dashboard.
2. In the upper right corner of the dashboard, click **Customize**, and then click **Add Widget**.



3. Specify a name for the widget in the **What Do You Want To Name This Widget** field.
4. Specify the resource in the **Which Resource Are You Interested In** field.
5. Choose one of the following display types for the **What Kind Of Display Do You Want** field.

- **Table display** The table display provides a general outline of the relative values. This display type provides sparklines—simple graphs that are good for showing trends and variations. The table display is the default display type.
- **Graph display** The graph display is more detailed and is useful for determining exact values at specific times. For example, you notice an unusual peak in a sparkline. To find out more information, such as when a peak occurred, add a widget that uses the graph display to display one or two metrics.

6. Follow the instructions in the **Add Widget** window to add a widget.

   For information about a specific metric, see "List of metrics" on page 1044.

7. Click **Create**.

To edit or delete a widget

- In the widget pane, click the dropdown arrow in the upper right corner of the widget, and then choose either:
  - **Settings** Edits the widget's settings.
  - **Delete** Deletes the widget.

See also

- "Dashboards" on page 1028
- "Overview dashboard" on page 1026

Closing connections from the Monitor

Any user can use the Monitor to view and close connections to database resources as long as the user knows the database's DBA user name and password.

To close a connection (Monitor)

1. Log in to the Monitor.
2. Open the dashboard for the SQL Anywhere resource.

   Choose **Dashboards** » *name of dashboard*.

3. In the **Connections** widget, select a connection, and then click the **x** beside the name of the connection.
4. When prompted enter the DBA user name and password of the database.

   When you close a connection, you must supply the DBA user ID and password for the database. These credentials are used to connect to the database and close the specified connection. The DBA credentials are then removed from the Monitor.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

5. Click **OK**.

See also

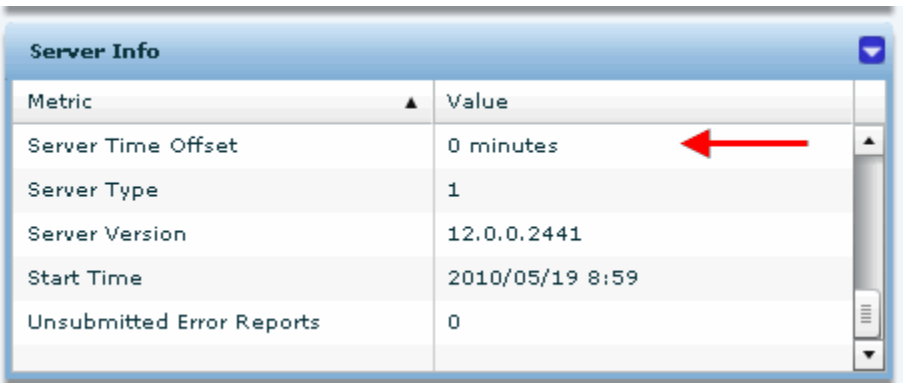
- "Lesson 5: Add a new dashboard and widgets" on page 1018

Understanding how time is displayed

All times displayed in the Monitor are based on the 24-hour clock and are local to the time on the computer that the Monitor is running on.

To find the time difference between the Monitor and your browser

1. Log in to the Monitor.
2. Open the dashboard for the resource.
3. In the **Server Info** widget, find the **Server Time Offset** metric.



The **Server Time Offset** records the time difference between the time on the computer that the Monitor is running and the time on the computer that you are using to view the Monitor data.

See also

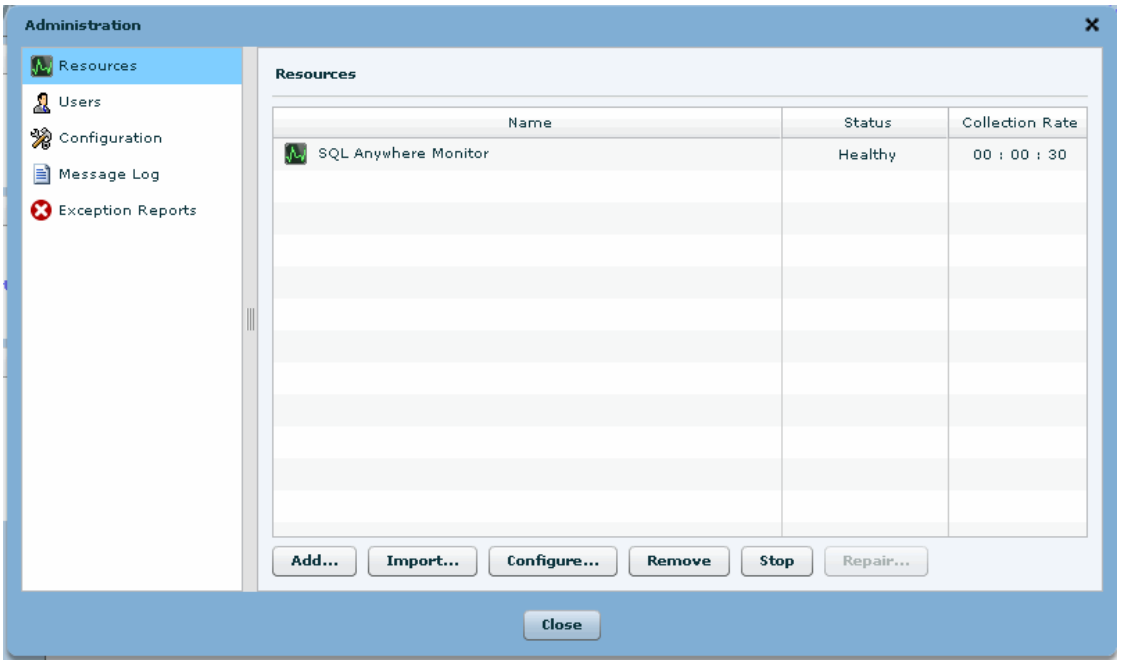
- "Metrics" on page 1042

Administration window

Note
Only administrators can access the **Administration** window. For information about administrators, see "Monitor users" on page 1048.

As an administrator, you can select the resources (for example, SQL Anywhere databases) that you want to monitor, and:

- Add, edit, and delete resources. See "Resources" on page 1033.
- Add and edit users. See "Monitor users" on page 1048.
- Configure the Monitor. See "Back up the Monitor" on page 1058.
- View the Message Log. See "Message Log" on page 1032.
- View the Exception Reports. See "Exception Reports" on page 1033.



See also

- "Metrics" on page 1042

Message Log

The **Message Log** contains informational messages from and about the SQL Anywhere Monitor regarding its operation. Messages are displayed in a table with the most recent message at the top.

To view the message log

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Click **Message Log**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Exception Reports

When the SQL Anywhere Monitor experiences a fatal error or a crash occurs, an exceptions report is created about what was happening at the time of the problem.

To view the exception reports

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Click **Exception Reports**.

Resources

A **resource** is a SQL Anywhere database, a MobiLink server, a MobiLink Server Farm, or a Relay Server farm. As an administrator, you add resources to the Monitor, and then you start monitoring them.

Monitoring of a resource starts:

- Automatically when the Monitor starts. When you start the Monitor, by default, it connects to the resources and collects metrics for *all* resources in the Monitor.
- Automatically when an administrator adds a resource. After a resource is added, the Monitor attempts to connect to the resource and starts monitoring it. See "Add resources" on page 1033.
- Automatically at the end of a blackout period. The Monitor automatically attempts to connect to the resource and resume monitoring.
- When an administrator opens the **Administration** window, clicks **Resources**, selects a resource from the list, and clicks **Start**.

SQL Anywhere Monitor resource

The default resource, named **SQL Anywhere Monitor**, reports on the health of the Monitor itself. This default resource is useful for monitoring the computer that the Monitor is running on and sending alerts when the Monitor is experiencing problems. You cannot modify this resource, nor can you stop monitoring it.

See also

- "Resource List widget" on page 1026

Add resources

To monitor a database, an administrator must first add the resource to the Monitor. By default, after the resource is added, monitoring starts.

As an administrator, you can add a resource one at a time or you can add multiple resources via an Import file. See "Add multiple resources" on page 1035.

To add a database resource to monitor

1. Log in to the Monitor as an administrator.
2. Click **Administration**.
3. Click **Resources**.
4. Click **Add**.
5. Follow the instructions in the **Add Resource** window to add a resource to monitor a database. In the **Host** field, specify the hostname or IP address of the computer on which the database server is running. You can use the name localhost to represent the current computer. This term is required. See "Host connection parameter" on page 291.

   If the database is part of a mirroring system and you always want to monitor the primary server in this system, then in the **Other** field, type **NODE=PRIMARY**. See "NodeType (NODE) connection parameter" on page 300.
6. Click **Create**.
7. When prompted, enter the DBA user ID and password for the database, and then click **OK**.

   When you add a database as a resource to be monitored, you must supply the DBA user ID and password for the database. These credentials are used to:

   - Connect to the database.
   - Create a new user named sa_monitor_user. The Monitor uses the sa_monitor_user to connect to the database and monitor it.
   - Install the database objects needed by the sa_monitor_user to monitor the database. For a list of the objects installed, see "Installed objects" on page 1060.
   - Discard from the Monitor the DBA credentials. Because the sa_monitor_user is added to the database being monitored, it is not necessary to store the DBA credentials anywhere outside the database that is being monitored. See "Installed objects" on page 1060.

   The resource is added and monitoring of the resource starts automatically.
8. Click **OK**.
9. Click **Close**.

   The resource appears in the **Overview** dashboard **Resource List**.
10. Optional: Add a dashboard for the resource. By default, when a resource is added, a dashboard is not.
    a. Open the **Overview** dashboard.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Click **Dashboards** » **Overview**.

b. In the **Resource List**, click the new resource.

The Monitor creates and opens a new dashboard for the resource. The name of the new dashboard appears in the **Dashboards** section of the sidebar.

See "Dashboards" on page 1028.

See also

- "Add multiple resources" on page 1035
- "Start the Monitor" on page 1021

Add multiple resources

As an administrator, you can add multiple resources to the Monitor by creating a comma separated values (CSV) file, and then importing the file.

To add multiple resources to the Monitor

1. Create a CSV file.

   Each line in the CSV file contains information about a single resource. Each comma-separated term within a line describes an attribute of the resource. The order of the terms is important. The following table describes the terms and their order:

   - **CSV file format for SQL Anywhere database resources**

Position in the line	Term	Description
1	Resource type	Type **sa** to Indicate that the resource is a SQL Anywhere database. This term is required.
2	Resource name	Specify the resource name that the database will have in the Monitor. This term is required.

Position in the line	Term	Description
3	Username	Specify the DBA user name used to connect to the database. This term is required.
4	Password	Specify the DBA password used to connect to the database. This term is required.
5	Host (SQL Anywhere)	Specify the hostname or IP address of the computer on which the database server is running. You can use the name localhost to represent the current computer. This term is required.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Position in the line	Term	Description
6	Port (SQL Anywhere)	Specify the port number on which the SQL Anywhere database server is running. The default port number for SQL Anywhere is 2638. This term is optional. The default value is 0
7	Server (SQL Anywhere)	Specify the name of the server that the database is connected to. This term is optional.
8	Database (SQL Anywhere	Specify the name of the database. This term is optional.

Position in the line	Term	Description
9	Connection Parameters (SQL Anywhere)	Specify additional connection parameters to use when connecting to the database. List the connection parameters as a semicolon delimited list of option=value pairs. The default value is "".This term is optional.

The following is an example of an import file that contains two SQL Anywhere database resources.

```
sa,demo,DBA,sql,localhost,demo12,demo
sa,demo3,DBA,sql,localhost,49152,demo12,demo
```

2. Log in as an administrator to the Monitor.
3. Click **Administration**.
4. Click **Resources**.
5. Click **Import**.
6. Locate the import file and click **Open**.

   The resources from the import file are added to the Monitor and monitoring starts.
7. Click **Close**.
8. Click **Close**.
9. The imported resource is added to the **Resource List** in the **Overview** dashboard.

Stop monitoring resources

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You stop monitoring resources when you do not want the Monitor to collect metrics from a SQL Anywhere database. For example, you want to stop monitoring when you know that the resource will be unavailable; otherwise, you receive alerts until the resource is available. Except for the default Monitor resource, you can stop monitoring any resource at any time.

When you stop monitoring a resource, the Monitor:

- Stops collecting metrics for the resource.
- Stops issuing alerts for the resource.

There are two ways to stop monitoring a resource:

- **Schedule a regular, repeating, blackout period** This method is a good choice when the following conditions apply:
  - You must repeatedly stop monitoring the database. For example, you perform regular maintenance at the end of each month.
  - You know in advance how long the database is unavailable. For example, you know that your regular maintenance takes four hours.
  - You need monitoring to automatically restart. When a blackout completes, the Monitor attempts to reconnect to the resource and to continue collecting data.

  To use this method, you create blackouts to make the Monitor stop monitoring at specified times. See "Automatically stop monitoring resources using blackouts" on page 1040.
- **Manually stop the monitoring** This method is a good choice when the following conditions are met:
  - You need to stop monitoring for infrequent or one-time tasks. For example, you need to stop monitoring because the computer that the resource is running on needs to be taken off-line for special maintenance.
  - You are available to restart the monitoring afterward. When a resource has been stopped manually, the Monitor waits for you to restart the monitoring.

  To use this method, see "Manually stop monitoring resources" on page 1039.

If you want to permanently stop monitoring a resource, you can remove it from the Monitor. See "Remove resources" on page 1041.

Manually stop monitoring resources

The following procedure describes how to manually stop a resource. For information about what happens when you stop a resource, see "Stop monitoring resources" on page 1038.

To manually stop a resource

1. Log in as an administrator to the Monitor.
2. Choose **Tools** » **Administration**.
3. Click **Resources**.
4. Select the resource, and then click **Stop**.
5. Click **Close**.

See also

- "Add resources" on page 1033
- "Add resources" on page 1033
- "Automatically stop monitoring resources using blackouts" on page 1040

Automatically stop monitoring resources using blackouts

Blackouts are times when you do not want the Monitor to collect metrics. When a blackout completes, the Monitor attempts to reconnect to the resources and to continue collecting data.

The following procedure describes how to stop a resource using blackouts. For information about what happens when you stop a resource and about when you should use blackouts, see "Stop monitoring resources" on page 1038.

Blackouts occur in the local time of the resource. See "Understanding how time is displayed" on page 1031.

To configure the blackout time

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Click **Resources**.
4. Select the resource, and then click **Configure**.
5. Click **Blackouts**.
6. Click **New**.
7. In the **New Blackout Period** window, specify the date and time (24 hour clock) for the blackout.

   The time is local to the computer where the resource database resides.
8. Click **Save**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

9. Click **Save**.
10. Click **OK**.
11. Click **Close**.

See also

- "Add resources" on page 1033
- "Manually stop monitoring resources" on page 1039

Repair database resources

Repairing a resource reinstalls the database objects needed to monitor the resource. The monitoring options are left unchanged. For a list of the objects installed, see "Installed objects" on page 1060.

As an administrator, you can repair only SQL Anywhere database resources. You cannot repair the default resource for the Monitor (named **SQL Anywhere Monitor**). Each time you repair a resource, you must specify the DBA user ID and password for the database.

To repair a SQL Anywhere resource

1. Log in to the Monitor as an administrator.
2. Click **Administration**.
3. Click **Resources**.
4. Select the database resource to repair.
5. If the resource is currently being monitored, click **Stop**.
6. Click **Repair**.
7. When prompted, type the DBA user ID and password for the SQL Anywhere database. The DBA credentials are used to connect to the database, and then they are removed from the Monitor.
8. Click **Repair**.
9. Click **OK**.
10. Click **Start** to restart monitoring the resource. See "Resources" on page 1033.
11. Click **Close**.

Remove resources

You should only remove resources when you are certain that you don't need to monitor them; for example, if the database is no longer being used.

Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Removing a resource causes the Monitor to:

- Permanently stop monitoring the resource.
- Discard the metrics collected for the resource.

When you remove a database resource, the Monitor does not delete the monitoring objects installed in the database. For information about deleting these objects, see "Deleting monitoring objects" on page 1061.

Only administrators can remove resources. You cannot delete the **SQL Anywhere Monitor** resource.

To remove a resource

1. Log in to the Monitor as an administrator.
2. Click **Administration**.
3. Click **Resources**.
4. Select the resource.
5. Click **Remove**.
6. Click **Yes**.
7. Click **Close**.

See also

- "Stop monitoring resources" on page 1038

Metrics

The Monitor collects and stores metrics from databases, including, but not limited to:

- Whether the resource is running.
- Whether the computer that the resource is running on is running properly and is connected to the network.
- Whether the resource is listening and processing requests.
- Whether there are any obvious problems such as long running queries or blocked users.

Collection intervals

Metrics displayed in the Monitor are only as precise as the collection interval. As an administrator, you can change the rate at which metrics are collected by the Monitor. By default, metrics are collected every 30 seconds.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To set the collection interval for a resource

1. Log in to the Monitor as an administrator.
2. Click **Resources**.
3. Select the resource, and then click **Configure**.
4. Click **Collection Interval**.
5. Specify the interval at which metrics are collected (hours:minutes:seconds). The collection interval must be at least 10 seconds long.
6. Click **Save**.
7. Click **OK**.
8. Click **Close**.

See also

- "Refresh metrics" on page 1043

Export metrics

You can export metrics that have a graph or table associated with them, to an XML file. For example, most of the metrics in the **Key Performance Metrics** widget can be exported.

To export metrics

1. Log in to the Monitor.
2. Open a dashboard.
3. On a widget that displays the metrics, click the dropdown menu arrow, and then click **Export**.
4. Follow the instructions in the **Export Metrics** window to export metrics.
5. Click **Export**.
6. When prompted, specify a file name.

   An XML file is created containing the specified metrics.

Refresh metrics

By default, the Monitor display is automatically refreshed every minute. You can change the refresh the interval by clicking **Tools** » **User Settings**. This setting is independent of the collection interval rate for a

resource, which specifies how often the Monitor collects metrics from the resource being monitored. See "Collection intervals" on page 1042 .

To set the refresh rate

1. Choose **Tools** » **User Settings**.
2. Change the settings as required. The default is one minute.
3. Click **OK**.

When you click **Refresh Data** the Monitor retrieves and displays the latest metrics.

To refresh metrics

- Click **Refresh Data**.

When you press F5, the Monitor reloads the browser, retrieves the metrics that the Monitor has collected to date, and displays the metrics.

To reload the Monitor

- Press F5.

List of metrics

List of metrics for SQL Anywhere resources

Metric name	Description
Active HTTP Requests	Returns the number of HTTP connections that are actively processing an HTTP request. An HTTP connection that has sent its response is not included. See the HttpNumActiveReq property in "Database server properties" on page 644.
Active HTTPS Requests	Returns the number of secure HTTPS connections that are actively processing an HTTPS request. An HTTPS connection that has sent its response is not included. See the HttpsNumActiveReq property in "Database server properties" on page 644.
Arbiter State	Shows one of the following values: • **Connected** The arbiter server is connected to the primary server. • **Disconnected** The arbiter server is not connected to the primary server. • **0** There is no arbiter server in this configuration.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Metric name	Description
Cache Dirty	The number of cache pages that are dirty (needing a write). See the CacheFileDirty property in "Database server properties" on page 644.
Cache Pinned	The number of pinned cache pages. See the CachePinned property in "Database server properties" on page 644.
Cache Replacements	The number of pages in the cache that have been replaced. See the CacheReplacements property in "Database server properties" on page 644.
Cache Size	The current cache size, in kilobytes. See the CurrentCacheSize property in "Database server properties" on page 644.
Connection Count	Shows the current number of connections to the database. See "sa_conn_info system procedure" [*SQL Anywhere Server - SQL Reference*].
CPU Usage	Shows the percentage of CPU time used by the SQL Anywhere server.
Database Name	Shows name of the database. See "Name database property" on page 671.
Disk Reads	Measures the rate at which data is being read from the disk (in kilobytes per second). This value is calculated based on the DiskRead property. See the DiskRead property in "Database server properties" on page 644.
Disk Writes	Measures the rate at which data being written to the disk (in kilobytes per second). This value is calculated based on the DiskWrite property. See the DiskWrite property in "Database server properties" on page 644.
File Size	Shows the size of the main database file. See "Predefined dbspaces" on page 15.
Free Disk Space	Shows the amount of free space on your disk. See "sa_disk_free_space system procedure" [*SQL Anywhere Server - SQL Reference*].
Host	Shows the name of the computer running the database server. Typically, this is the computer's host name. See the MachineName property in "Database server properties" on page 644.
HTTP Connections	Returns the number of HTTP connections that are currently open within the database server. They may be actively processing a request or waiting in a queue of long lived (keep-alive) connections. See the HttpNumConnections property in "Database server properties" on page 644.
HTTP Sessions	Returns the number of active and dormant HTTP sessions within the database server. See the HttpNumSessions property in "Database server properties" on page 644.

Metric name	Description
HTTPS Connections	Returns the number of HTTPS connections that are currently open within the database server. They may be actively processing a request or waiting in a queue of long lived (keep-alive) connections. See the HttpsNumConnections property in "Database server properties" on page 644.
Last Checked Time	Returns the last time the Monitor collected data for the resource.
License Type	Returns the license type. Can be networked seat (per-seat) or CPU-based. See the LicenseType property in "Database server properties" on page 644.
Licensed Company	Shows the name of the licensed company. See the CompanyName property in "Database server properties" on page 644.
Licensed Seats	Shows the number of licensed seats or processors. See the LicenseCount property in "Database server properties" on page 644.
Licensed User	Shows the name of the licensed user. See the LicensedUser property in "Database server properties" on page 644.
Long Running Queries	Lists queries that exceed the specified long running query threshold.
Main Heap Pages	The number of pages used for global server data structures. See the MainHeapPages property in "Database server properties" on page 644.
Max Cache Size	The largest value the cache has reached in the current session, in kilobytes. See the PeakCacheSize property in "Database server properties" on page 644.
Mirror Mode	Shows **Mirroring Is Not Enabled On This Database** if database mirroring is not in use. If mirroring is enabled, shows **Synchronous** if the mirroring mode specified with the -xp command line option is synchronous, and **Asynchronous** otherwise.
Mirror State	Returns one of the following values: • **Synchronizing** The mirror server is not connected or has not yet read all the primary server's log pages. This value is also returned if the synchronization mode is asynchronous. • **Synchronized** The mirror server is connected and has all changes that have been committed on the primary server.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Metric name	Description
Operating System	Shows the operating system on which the software is running. See the Platform property in "Database server properties" on page 644.
Operating System Version	Shows the operating system on which the software is running, including build numbers and service packs. See the PlatformVer property in "Database server properties" on page 644.
Partner State	Shows one of the following values: • **Connected** The mirror server is connected to the primary server. • **Disconnected** The mirror server is not connected to the primary server.
Peak Cache Size	The largest value the cache has reached in the current session, in kilobytes. See the PeakCacheSize property in "Database server properties" on page 644.
Processor Architecture	Shows a string that identifies the processor type. See the ProcessorArchitecture property in "Database server properties" on page 644.
Queries Processed	Shows the rate (queries/second) at which queries are processed. See the QueryOptimized, QueryReused, and QueryBypassed properties in "Database properties" on page 659.
Seat Count	Shows the number of unique client network addresses connected to a network database server. See the UniqueClientAddresses property in "Database server properties" on page 644.
Server Language	Shows the locale language, which is the language that is expected to be used by users of the database server. See "Understanding the locale language" on page 417.
Server Name	The name of the database server for the current connection. See "ServerName (Server) connection parameter" on page 306.
Server Time Offset	Records the time difference between the time on the computer that the Monitor is running and the time on the computer that you are using to view the Monitor data. See "Understanding how time is displayed" on page 1031.
Server Type	Shows the type of database server being monitored. Values include Personal and Network.
Server Version	Shows the version of the software being run. See the ProductVersion property in "Database server properties" on page 644.

Metric name	Description
Start Time	Shows the time when the SQL Anywhere database started.
Total Disk Space	Shows the size of your disk.
Unavailable Since	Shows the time since the resource became unavailable.
Unscheduled Requests	Shows the number of unscheduled requests. See "UnschReq server property" on page 659.
Unsubmitted Error Reports	Shows the number of unsubmitted error reports for the database. An error report is submitted when SQL Anywhere software crashes. See "Suppress alerts for unsubmitted error reports from resources" on page 1055.

Monitor users

You must log in to the Monitor with a user name and password. The user name and password for logging in to the Monitor are case sensitive.

Default user

By default, when you first start the Monitor, it has one administrator user, named **admin**, with the password **admin**. By default, this user has full permissions. It is recommended that you change the default administrator password to restrict access to the Monitor. See "Edit Monitor users" on page 1051.

The Monitor supports three types of users:

User type	Description
Read-only user	Has read-only access to monitor resources. Read-only users can view the metrics, but cannot access the **Administration** window. When you create a user from the log-in screen, this user is a Read-only user. A user name and password are required.
Operator	Has read-only access to monitor resources and can receive alerts. These users can view the metrics, can receive email alerts, and can resolve and delete alerts. However, operators cannot access the **Administration** window. A user name and password are required.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

User type	Description
Administrator	Has the same access as an operator, and can also configure resources and add users. Administrators can also access the **Administration** window. The default user, **admin**, is an administrator. A user name and password are required.

All users, once they have logged in, can check their user type and change their settings. However, only an administrator can change a user's type. See "Edit Monitor users" on page 1051.

To check your Monitor user type

1. Log in to the Monitor.
2. Choose **Tools** » **User Settings**.
3. Review the **User Type** setting in the window.

See also

- "Administration window" on page 1031

Create Monitor users

By default, from the log-in screen, anyone can create their own read-only user and have read-only access to the Monitor. However, administrators can change this default so that only administrators can create users; see "Only allow administrators the ability to create users" on page 1052.

To create a read-only user from the log-in screen

1. Click **Create New User**.

   Note
   If the **Create New User** link is not available, then the administrator has changed the default behavior so that only administrators can create new users. Contact your Monitor administrator to create a new user.

2. Fill in the information for the new user.

   An email address is only required if you want to receive email alerts from the Monitor. Contact your Monitor administrator to change your user type to operator or administrator, and sign you up to receive email alerts. See "Send alert emails" on page 1057.

When logged in, administrators can create any type of user, including other administrators.

To create new users when logged-in as an Administrator

1. Log in to the Monitor as an administrator.

2. Choose **Tools** » **Administration**.
3. Click **Users**.
4. Click **New**.
5. Fill in the information for the new user. An email address is only required for users who should receive email alerts from the Monitor.
6. Select a language from the **Preferred Language Type** dropdown menu. The specified language sets the language used by the Monitor, including the language used in alerts.
7. Select a user type. Each type has different permissions. See "Monitor users" on page 1048.
8. Click **Next** to create the resource dashboards for this user.
9. Click **Save**.
10. Click **Close**.
11. If you created an operator or an administrator user, this user can receive alert notifications by email. See "Send alert emails" on page 1057.

See also

- "Edit Monitor users" on page 1051
- "Only allow administrators the ability to create users" on page 1052

Associate Monitor users with resources

You must associate an operator or administrator user with a resource if you want the user to receive email alerts about the associated resource.

To associate an operator or administrator with a resource

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Ensure that the operator or administrator has an email address specified in their user account.

   Click **Users** and verify that the user has an email address specified.
4. Click **Resources**, select a resource from the list, and then click **Configure**.
5. Click **Operators**.
6. Select a user from the **Available Operators** list and then click **Add**.

   The user name appears in the **Selected Operators** list.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

7. Click **Save**.
8. Click **OK**.
9. Click **Close**.
10. Ensure that the Monitor is set up to send alert notifications by email. See "Enable the Monitor to send alert emails" on page 1057.

See also

- "Send alert emails" on page 1057

Edit Monitor users

All users, once they have logged in, can change their user settings by clicking **Tools** » **User Settings**. However, only an administrator can change a user's type.

To edit an existing Monitor user

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Click **Users**.
4. Select the user to edit, and then click **Edit**.
5. Change the settings for the user as required. An email address is only required for users who should receive email alerts from the Monitor.
6. Click **Save**.
7. Click **Close**.
8. If you created an operator or an administrator user, this user can receive alert notifications by email. See "Send alert emails" on page 1057 and .

See also

- "Create Monitor users" on page 1049
- "Delete Monitor users" on page 1051

Delete Monitor users

Deleting a user removes the user from the Monitor and disassociates the user from any resource.

To delete an existing Monitor user

1. Log in to the Monitor as an administrator.

2. Choose **Tools** » **Administration**.
3. Click the **Users**.
4. Select the user, and then click **Delete**.
5. Click **Yes** to delete the selected user.

   The user is deleted from the Monitor.
6. Click **Close**.

See also

- "Create Monitor users" on page 1049
- "Edit Monitor users" on page 1051

Only allow administrators the ability to create users

By default, from the log-in screen, anyone can create their own user name and password and have read-only access to the Monitor. However, as an administrator, you turn off this feature so that only administrators can create users.

To restrict user creation to administrators

1. Log in to the Monitor as an administrator.
2. Chose **Tools** » **Administration**.
3. Click **Configuration**.
4. Click **Edit**.
5. Click **Options**.
6. Clear the **Allow Anyone Read-only Access To The SQL Anywhere Monitor** option.
7. Click **Save**.
8. Click **Close**.

See also

- "Create Monitor users" on page 1049
- "Edit Monitor users" on page 1051

Alerts

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

An **alert** is a condition or state of interest about a resource that should be brought to an administrator's or operator's attention. Alerts are detected by the Monitor based on metrics that are collected. They are not detected at the database being monitored.

There are predefined alerts for conditions such as low disk space, failed login attempts, and high memory usage. You can change the default threshold values by editing the resource. See "Specify alert thresholds" on page 1054.

When an alert condition is met, the alert is listed in the **Alert List** widget for the specified resource. See "Alerts List widget" on page 1027.

By default, alerts appear in the **Alert List** widgets and they include information about the cause of the problem, and provide advice for resolving the problem. In the **Resource List** the resource's status changes to reflect the existence and severity of the alert. See "Overview dashboard" on page 1026.

You can configure the Monitor to send an email to operators and administrators when an alert occurs. See "Send alert emails" on page 1057.

View alerts

The Monitor keeps only the most recent 50 alerts in the alert list.

> **Note**
> Any logged-in user can view alerts; however, only operators and administrators can resolve and delete alerts.

To view an alert

1. Choose **Alerts** » **Today**.
2. Select an alert from the list, and then click **Details**.
3. Click **OK**.

See also

- "Resolve alerts" on page 1053
- "Delete alerts" on page 1054
- "Send alert emails" on page 1057

Resolve alerts

As an operator or an administrator, you can mark an alert as resolved after the issue that triggered the alert has been addressed.

Resolving an alert causes the Monitor to change the alert **Status** to **Resolved by *user-name***, but leaves the alert in the alert list. If you want to remove the alert from the list, you must delete it. See "Delete alerts" on page 1054.

To resolve an alert

1. Log in to the Monitor as an administrator or operator user.
2. Click **Overview**.
3. In the **Alerts List** widget, select an alert from the list and click **Mark Resolved** to resolve the selected alert.

   The value in the **Status** column changes to **Resolved by *your-user-name***.

   If this alert was the resource's only unresolved alert, the resource's status in the **Resource List** widget changes to Healthy (no icon is present).

See also

- “Overview dashboard” on page 1026
- “Delete alerts” on page 1054
- “Send alert emails” on page 1057
- “View alerts” on page 1053
- “Alerts” on page 1052

Delete alerts

As an operator or an administrator, you can delete any alert from an **Alerts List**. You can delete alerts, regardless of their statuses.

To delete alerts

1. Log in to the Monitor as an administrator or operator user.
2. Click **Overview**.
3. In the **Alerts List** widget, select an alert from the list and click **Delete**.

   The alert is removed from the alerts list.

See also

- “Resolve alerts” on page 1053
- “Send alert emails” on page 1057
- “View alerts” on page 1053
- “Alerts” on page 1052

Specify alert thresholds

As an administrator you can configure when alerts should be issued. But, you cannot configure what metrics the Monitor collects nor can you configure the default resource, the SQL Anywhere Monitor.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To configure when alerts should be issued

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Select **Resources**, and then select a resource from the list.
4. Click **Configure**.
5. Click **Alert Thresholds**. Edit the thresholds. For definitions of the alerts, see "Alert thresholds" on page 1056.
6. Configure the other settings as required.
7. Click **Save**.
8. When you are prompted for the required authorization, specify the database's DBA user ID and password.

   When you edit the resource for a SQL Anywhere database, you must supply the DBA user ID and password for the database. These credentials are used to connect to the database and alter the installed database objects needed to monitor it. The DBA credentials are then removed from the Monitor.
9. Click **OK**.
10. Click **Close**.

See also

- "Metrics" on page 1042
- "Alert thresholds" on page 1056

Suppress alerts for unsubmitted error reports from resources

As an administrator, you can configure whether the Monitor sends out alerts when resources have unsubmitted error reports. By default, the Monitor does not send these alerts. For information about error reports and about how to submit them, see "Error reporting in SQL Anywhere" on page 996.

To suppress alerts for unsubmitted error reports

1. Log in to the Monitor as an administrator.
2. Choose **Tools** » **Administration**.
3. Click **Configuration**.
4. Click **Edit**.
5. Click **Options**.

6. Select **Suppress unsubmitted error report alerts form resources**.
7. Click **Save**.
8. Click **Close**.

Alert thresholds

As an administrator, you can configure the thresholds that are used to trigger alerts. See "Specify alert thresholds" on page 1054.

The following list describes the alerts and their default thresholds.

Alert thresholds for SQL Anywhere database resources

- **Alert When CPU Usage Exceeds The Given Threshold For The Given Number of Seconds**
  - **Threshold (%)** Issue an alert when the resource's CPU use reaches the specified percentage. The default is 95 percent.
  - **Seconds** The default is 30 seconds.
- **Alert When Memory Usage Reaches *X*% Of The Maximum Cache Size** Issue an alert when the memory usage of the resource reaches the specified percentage. The default is 90 percent.
- **Alert When Free Disk Space Per Dbspace Is Less Than *X* MB On The Disk** Issue an alert when the free disk space per dbspace is less than the specified amount. The default is 100 MB.
- **Alert When A Connection Has Been Blocked For Longer Than X Seconds** Issue an alert when a connection has been blocked for longer than the specified time. The default is 10 seconds.
- **Alert When The Number Of Connections In Use Reaches *X* % Of The License Limit** Issue an alert when the number of connections in use reaches the specified percentage of the license limit. The default is 85 percent.
- **Alert When A Query Has Run For Longer Than *X* Seconds** Issue an alert when a query has run for longer than the specified time. The default is 10 seconds.
- **Alert When The Number Of Unscheduled Requests Reaches *X*** Issue an alert when the number of unscheduled requests reaches the specified amount. The default is 5 requests.
- **Alert When It Has Been More Than The Given Number Of Days Since The Last Backup** Issue an alert when the number of days since the resource database was last backup exceeds the specified number of days. The default is 14 days.
- **Suppress Alerts For The Same Condition That Occur Within *X* Minutes** This option prevents you from receiving duplicate alerts within a specified time. The default is 30 minutes.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Send alert emails

As an administrator, you can configure the Monitor to send an email to specified operators and administrators when an alert occurs for specified resources.

To have the Monitor send alert notifications by email

1. Log in to the Monitor as an Administrator.
2. Create an administrator or operator with an email address. See "Create Monitor users" on page 1049.
3. Associate the administrator or operator with a resource. See "Associate Monitor users with resources" on page 1050.
4. Enable the Monitor to send emails. See "Enable the Monitor to send alert emails" on page 1057.

Enable the Monitor to send alert emails

As an administrator, you can configure the Monitor to send emails to operators and administrators when alerts occur. The Monitor supports the SMTP and MAPI protocols for sending emails.

To enable the Monitor to send alert notifications by email

1. Log in to the Monitor as an Administrator.
2. Choose **Tools** » **Administration**.
3. Click **Configuration**.
4. Click **Edit**.
5. Click **Alert Notification**.
6. Select **Send Alert Notifications By Email**.
7. Choose either SMTP or MAPI for the **Which Protocol Do You Want To Use To Send Alerts By Email?** field.
8. Configure the other settings as required.
   - **MAPI**
     - **User Name** Type the user name for the MAPI server.
     - **Password** Type the password for the MAPI server.
   - **SMTP**
     - **Server** Specify which SMTP server to use. Type the server name or the IP address for the SMTP server. For example, *SMTP.yourcompany.com*.
     - **Port** Specify the port number to connect to on the SMTP server. The default is 25.

- **Sender Name** Specify an alias for the sender's email address. For example, *JoeSmith*.
- **Sender Address** Specify the email address of the sender. For example, *jsmith@emailaddress.com*.
- **This SMTP Server Requires Authentication** Select this option if your SMTP server requires authentication.
  - **User Name** Specify the user name to provide to SMTP servers requiring authentication.
  - **Password** Specify the password to provide to SMTP servers requiring authentication.

9. Click **Send Test Email**.
10. When prompted, enter an email address to send the test email to and click **OK**.

    A test email is sent to the email address specified.
11. Click **Save**.
12. Click **Close**.

    When an alert occurs, the Monitor sends alert emails to operators and administrators who have specified email addresses in their user accounts. These users receive emails for the resources that they are associated with. See "Create Monitor users" on page 1049 and "Associate Monitor users with resources" on page 1050.

See also

- "Resolve alerts" on page 1053
- "Delete alerts" on page 1054
- "View alerts" on page 1053

Back up the Monitor

By default, the Monitor performs maintenance on the metrics once a day at midnight. Maintenance affects metrics, not alerts.

As an administrator, you can:

- Schedule the Monitor to back up metrics.
- Control the amount of disk space that the Monitor uses.
- Perform maintenance on demand.

To back up metrics and control the amount of disk space used to store metrics

1. Log in to the Monitor as an Administrator.
2. Click **Administration**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

3. Select **Configuration**, and then click **Edit**.

4. Click **Maintenance**.

5. Specify a time (24-hour clock) when the Monitor should perform maintenance. By default, it performs maintenance at midnight. The time is local to the computer where the Monitor is running. See "Understanding how time is displayed" on page 1031.

6. Specify a directory where the Monitor should save the backed up data. The directory must exist on the computer where the Monitor is running.

7. Customize the **Data Reduction** settings:
   - **Reduce Metrics To A Representative Daily Value For Metrics Older Than** When you select this option, an average is taken for all numeric metrics that are older than the specified number of days, and then the numeric metrics are deleted. Non-numeric metrics are not deleted.
   - **Delete Values Older Than** When you select this option, all metrics that are older than the specified length of time are deleted.
   - **Delete Old Metrics When The Total Disk Space Used By The SQL Anywhere Monitor Becomes Greater Than (MB)** When you select this option, you specify the maximum amount of space that can be used to store the metrics. When the amount of disk space used reaches or exceeds the amount specified, the Monitor deletes metrics, starting with the oldest metrics. Metrics are deleted until enough free space exists to store new metrics.

8. Click **Perform Maintenance Now** to run the backup immediately. When prompted, click **OK**.

9. Click **Save**.

10. Click **Close**.

The following procedure describes how to replace the Monitor database with a backup copy.

To restore a backup copy of the Monitor database

1. Stop the Monitor, if it is running.

2. From your backup directory, copy the *samonitor.db* database file and *samonitor.log* log file.

3. Paste these files into the directory where the current Monitor database and log file are located. When prompted, overwrite the existing files.

   The default locations of the version 12.0.0 Monitor database files are listed in the following table:

Operating system	Monitor directory
Windows XP (installed with SQL Anywhere)	`C:\Documents and Settings\All Users\Documents\SQL Anywhere 12\Monitor\samonitor.db`

Operating system	Monitor directory
Windows XP (installed on a separate computer)	`C:\Documents and Settings\All Users\Documents\SQL Anywhere 12 Monitor\samonitor.db`
Windows Vista (installed with SQL Anywhere)	`C:\Users\Public\Documents\SQL Anywhere 12\Monitor\samonitor.db`
Windows Vista (installed on a separate computer)	`C:\Users\Public\Documents\SQL Anywhere 12 Monitor\samonitor.db`
Linux (installed with SQL Anywhere)	`/opt/sqlanywhere12/samonitor.db`
Linux (installed on a separate computer)	`/opt/sqlanywhere12/samonitor.db`

4. Re-start the Monitor.

Installed objects

The following table lists the objects that are installed when you add a SQL Anywhere database as a resource to be monitored.

Object name	Object type	Description
sa_monitor_user	Database user	This read-only user is added to the database to collect metrics. Because this user is added to the database being monitored, it is not necessary to store the DBA credentials anywhere outside the database that is being monitored. It may be necessary to allow sa_monitor_user to bypass password verification. The sa_monitor_user has a random password known only to the Monitor and it does not have administrator privileges. S
sa_monitor_connection_failure	Table	This table contains metrics about failed connection attempts, and is used with sa_monitor_connection_failed_event. The metrics in this table are deleted as metrics are retrieved from the Monitor.
sa_monitor_connection_failed_event	Event	This event fires on the ConnectFailed system event (every time a connection attempt fails), and inserts a record into the sa_monitor_connection_failure table.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Object name	Object type	Description
sa_monitor_count_unsubmitted_crash_reports	Function	This function calls the xp_srvmon_count_unsubmitted_crash_reports procedure to gather a count of the number of unsubmitted crash reports.

Deleting monitoring objects

Because the database objects are owned by a single owner, you can delete all of them by executing the following statement:

```
DROP USER sa_monitor_user;
```

Reinstalling monitoring objects

To reinstall the database objects, see "Repair database resources" on page 1041.

Installing the SQL Anywhere Monitor in a production environment

It is recommended, particularly in production environments, that you:

1. Install and run the Monitor Production Edition.

   Advantages to running the Monitor Production Edition include:

   - The Monitor runs in the background as a service.
   - The Monitor starts automatically when the computer starts.
   - Upgrades and updates of SQL Anywhere do not overwrite or affect the Monitor Production Edition. In contrast, upgrades and updates of SQL Anywhere can affect the Monitor Developer Edition as it uses the installed SQL Anywhere on the back-end.

2. Install the Monitor on a computer that is different from the computer where the resources are running.

   Advantages to running the Monitor on a separate computer include:

   - The impact on the database server or other applications is minimized.
   - Monitoring is not affected if something happens to the computer where the resources are installed.

These instructions explain how to install the SQL Anywhere Monitor Production Edition.

To install the Monitor Production Edition (Windows)

1. Run the *setup.exe* file from the *Monitor* directory on your installation media, and follow the instructions provided.

On Windows, the Monitor service is started by the installation..

2. Open the default URL for logging in to the Monitor: *http://localhost:4950*.

> **Note**
> If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 1025.

3. Log in.

   When prompted, enter your user name and password for the Monitor. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 1048.

To install the Monitor Production Edition (Linux)

1. As the root user, run the *setup.tar* file from the *Monitor* directory on your installation media, and follow the instructions provided.

> **Note**
> On Linux, the Monitor Production Edition can only be run by the root user.

2. On Linux, by default, the Monitor Production Edition automatically starts the Monitor service.

3. Open the default URL for logging in to the Monitor: *http://localhost:4950*.

> **Note**
> If you are accessing the Monitor over a network, browse to **http://***computer-name***:4950**, where *computer-name* is the name of the computer the Monitor is running. See "Log in remotely to the Monitor" on page 1025.

4. Log in.

   When prompted, enter your user name and password for the Monitor. The default user is an administrator with the name **admin** and the password **admin.** See "Monitor users" on page 1048.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Upgrading the Monitor and migrating resources and metrics

Caution
Uninstalling the Monitor removes the application, as well as the resources and collected metrics.

If you want to preserve your current Monitor resources and metrics, you must:

1. Install a new version of the Monitor.
2. Migrate the resources and metrics.
3. Uninstall the older version of the Monitor.

See "Upgrading the SQL Anywhere Monitor and migrating resources and metrics" [*SQL Anywhere 12 - Changes and Upgrading*].

Securing the Monitor

You can secure communications between both the Monitor and your browser, and between the Monitor and resources it monitors.

Securing communications between the Monitor and your browser using transport-layer security (TLS)

You can use transport-layer security (TLS) to secure communication between the Monitor and your browser. The Monitor runs a web server that supports HTTPS connections using SSL version 3.0 and TLS version 1.0.

To set up TLS security for the Monitor

1. Obtain digital certificates from a certificate authority or create self-signed certificates with the Certificate Creation utility (createcert). See "Certificate Creation utility (createcert)" on page 775.
2. Alter the Monitor start line string to use the certificates.
3. Configure your browser to accept your new certificates, if required.

For more information, see http://www.sybase.com/detail?id=1063938.

Securing connections between the Monitor and your resources

You can use ECC or FIPS to encrypt communications between the Monitor and the resources it monitors. ECC encryption and FIPS-certified encryption require a separate license.

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

Troubleshooting the Monitor

In addition to the recommendations listed below, Administrators can use the Message Log and Exception Reports features to troubleshoot the Monitor. See "Message Log" on page 1032 and "Exception Reports" on page 1033.

Problem	Recommendation
When you press F5 to refresh the browser window, you are required to log in to the Monitor.	Enable JavaScript in your browser.
You receive a network communication error when you try to log in to the Monitor.	Start the Monitor. See "Start the Monitor" on page 1021.
After upgrading to the latest version of Adobe Flash Player you continue to receive instructions to upgrade Adobe Flash Player.	Verify that the installed version Adobe Flash Player is supported by your operating system. The Monitor is backwards compatible with version 10 of Adobe Flash Player. To determine the correct version, see http://www.adobe.com/products/flashplayer/systemreqs/.
The Monitor is unable to start monitoring a SQL Anywhere database resource.	Verify that the resource's password verification functions and login procedures allow the user sa_monitor_user to connect to the resource.
You are not receiving any alert emails.	Verify that the Monitor is properly configured to send emails and send a test email. See "Enable the Monitor to send alert emails" on page 1057. Verify that the alert emails from the Monitor are not being blocked by a virus scanner. See "xp_startsmtp system procedure" [*SQL Anywhere Server - SQL Reference*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Problem	Recommendation
The number of unscheduled requests reported by the Monitor appears to be less than the actual number of unscheduled requests.	When collecting metrics about the number of unscheduled requests, the Monitor executes query on the resource. This query could be an unscheduled request. Unscheduled queries are processed sequentially as they arrive. Therefore, if there are unscheduled requests when the Monitor attempts to execute its query, then this query must wait for the existing unscheduled requests to complete before it can execute. As a result, when the Monitor collects the number of unscheduled requests, this number does not include the unscheduled requests that existed between the time when the Monitor issued its query and the query executed.
You are not receiving alerts when the database disk space surpasses the specified threshold.	Between Monitor collection intervals, it is possible for a database to exceed the specified disk space alert threshold and the amount of space available. In such a case, the database would stop responding before the Monitor could collect the disk usage metrics and issue an alert. If your database grows quickly, set the disk space alert threshold to a higher number so that you can receive an alert before the database runs out of space. See "Alert thresholds" on page 1056.
You can't see the **Administration** window when you are logged into the Monitor.	You must be logged in to the Monitor as an administrator to have access to the **Administration** window. See "Monitor users" on page 1048.
You uninstalled the Monitor before migrating your data and now you have lost your resources and metric data.	Uninstalling the Monitor removes the application, as well as the resources and collected metrics. When upgrading, you need to install the new version of the Monitor, migrate your data, and then uninstall the old version. However, if you regularly backed up your Monitor, you can use the Migrate utility to migrate the data from the backed up files to the new version of the Monitor. See "Back up the Monitor" on page 1058.

Problem	Recommendation
When monitoring a database that is part of database mirroring system, you receive errors about not being able to modify a read-only database.	When you created the resource for your database, the database was the primary server in the database mirroring system. Now, a role switch has occurred and your database has assumed the role of the mirror server and it can only be accessed in read-only mode. To always monitor the primary server in a mirroring system, ensure that the Monitor uses the NODE connection parameter when connecting to the database. In the **Administration** window, select the mirrored database resource and click **Edit**. In the **Other** field, type **NODE=PRIMARY**. See "NodeType (NODE) connection parameter" on page 300.

The SQL Anywhere SNMP Extension Agent

If you are running SQL Anywhere on Windows (32-bit versions), you can use the SQL Anywhere SNMP Extension Agent in conjunction with SNMP management applications to manage your SQL Anywhere databases. One agent can be used to monitor several different databases running on different database servers running on different computers.

Using the SQL Anywhere SNMP Extension Agent, you can:

- Retrieve the value of all server and database statistics.
- Retrieve the value of all server and database properties.
- Retrieve the value of all PUBLIC database options.
- Set the value for any PUBLIC database option.
- Execute stored procedures.
- Generate traps based on property or statistic values.

Supplied files

The following files for the SQL Anywhere SNMP Extension Agent are included in your SQL Anywhere installation:

- **dbsnmp12.dll** The SQL Anywhere SNMP Extension Agent. This file is located in *install-dir\bin32*.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **iAnywhere.mib** The SQL Anywhere MIB contains all the OIDs for database server and database properties, statistics, and options that can be accessed using the SQL Anywhere SNMP Extension Agent.
- **RDBMS-MIB.mib** This is a generic MIB for relational database management systems and contains OIDs that can be accessed using the SQL Anywhere SNMP Extension Agent.
- **SNMPv2-SMI.mib** This MIB is referenced by the SQL Anywhere and RDBMS MIBs.
- **SNMPv2-TC.mib** This MIB is referenced by the SQL Anywhere and RDBMS MIBs.
- **SYBASE-MIB.mib** The Sybase MIB. This MIB is referenced by the SQL Anywhere MIB.
- **sasnmp.ini** This file lists the databases that the SQL Anywhere SNMP Extension Agent monitors. By default, this file is located in *install-dir\bin32*.

Understanding SNMP

Simple Network Management Protocol (**SNMP**) is a standard protocol used for network management. SNMP allows **managers** and **agents** to communicate: managers send requests to agents, and agents respond to queries from managers. Additionally, agents can notify managers when specific events occur using notifications called **traps**.

SNMP agents handle requests to get and set the values of variables for managed objects. Each variable has a single value, and values are generally strings or integers, although they may also be other types.

Variables are kept in a global hierarchy, and each variable has a unique number under its parent. The full name of a variable (including all its parents) is called the **Object Identifier** (OID). All OIDs that are owned by Sybase begin with **1.3.6.1.4.1.897**.

The list of OIDs that an agent supports, including their names, types, and other information are stored in a file called a **Management Information Base** (MIB).

A MIB is a database that stores network management information about managed objects. The MIB is separate from the SQL Anywhere database you are monitoring using the SQL Anywhere SNMP Extension Agent. The values of MIB objects can be changed or retrieved using SNMP. MIB objects are organized in a hierarchy with the most general information about the network located at the top level of the hierarchy. The SQL Anywhere SNMP Extension Agent supports the following MIBs:

- **SQL Anywhere MIB** A MIB created specifically for the SQL Anywhere SNMP Extension Agent. All the OIDs in the SQL Anywhere MIB begin with **1.3.6.1.4.1.897.2**. The SQL Anywhere MIB lists the OIDs for the statistics, properties, and option values that can be retrieved, and sometimes set, using the SQL Anywhere SNMP Extension Agent. See "The SQL Anywhere MIB" on page 1068.
- **RDBMS MIB** A generic, vendor-independent MIB for relational databases. This MIB contains information about the database servers and databases in your system. See "The RDBMS MIB" on page 1070.

The SQL Anywhere MIB

The SQL Anywhere MIB was created for the SQL Anywhere SNMP Extension Agent. It includes all database server statistics and properties, and all database statistics, properties, and options. The statistics and properties are all read-only (with a few exceptions), and the database options are all read-write.

By default, the SQL Anywhere MIB is located in *install-dir\snmp\iAnywhere.mib*.

For more information about the tables in the SQL Anywhere MIB, see "SQL Anywhere MIB reference" on page 1078.

For more information about setting values in the SQL Anywhere MIB, see "Setting values using the SQL Anywhere SNMP Extension Agent" on page 1075.

The following hierarchy describes the SQL Anywhere MIB:

OID	Name	Description
1.3.6.1.4.897.2.1.1.*n.db*	saServer.saSrvStat	Returns the value of server statistic *n* on database *db*.
1.3.6.1.4.897.2.1.2.*n.db*	saServer.saSrvProp	Returns the value of server property *n* on database *db*.
1.3.6.1.4.897.2.2.1.*n.db*	saDb.saDbStat	Returns the value of database statistic *n* on database *db*.
1.3.6.1.4.897.2.2.2.*n.db*	saDb.saDbProp	Returns the value of database property *n* on database *db*.
1.3.6.1.4.897.2.2.3.*n.db*	saDb.saDbOpt	Returns the value of database option *n* on database *db*.
1.3.6.1.4.897.2.3.1	saAgent.saVersion	Returns the version of the SQL Anywhere Extension Agent.
1.3.6.1.4.897.2.3.2.*db*	saAgent.saDbConnStr	Returns the connection string for database *db*.
1.3.6.1.4.897.2.3.3.*db*	saAgent.saConnected	Returns whether the SQL Anywhere Extension Agent is connected to database *db*. Setting this value to 0 causes the SQL Anywhere Extension Agent to disconnect from the database, while setting this value to 1 causes the SQL Anywhere Extension Agent to attempt to connect to the database.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Name	Description
1.3.6.1.4.897.2.3.4.*db*	saAgent.saStarted	Returns whether database *db* is running. Setting this value to 0 causes the SQL Anywhere Extension Agent to shut down the database[1], while setting this value to 1 attempts to start the database[2].
1.3.6.1.4.897.2.3.5.*db*	saAgent.saProc	Setting this value to a string *proc_name* causes the SQL Anywhere Extension Agent to executed the procedure *proc_name* in the database. Arguments can be supplied (for example, proc_name('string', 4)); if no arguments are supplied, parentheses **()** are appended to the name. Getting the value returns "".
1.3.6.1.4.897.2.3.6	saAgent.saRestart	Setting the value of this variable to 1 causes the agent to restart itself (it disconnects from all databases and reloads the *.ini* file). Getting the value returns 0.
1.3.6.1.4.897.2.3.7	saAgent.saInifile	Returns the full path of the *sasnmp.ini* file the SQL Anywhere Extension Agent is using.
1.3.6.1.4.897.2.4	saMetaData	Several virtual tables; each row represents a variable supported by the SQL Anywhere MIB.

[1] When stopping a database by setting this variable, the stop is unconditional, meaning that the database will be stopped even if it has active connections.

[2] To be able to start a database by setting this variable, the DBF parameter must be specified in the connection string (including the DBN, and DBKEY if it is required), and either the UtilDbPwd field must be set in the *sasnmp.ini* file, or the start database permission on the server (specified with the -gd server option) must be set to all.

saMetaData tables

The SQL Anywhere MIB includes metadata tables that provide a way to query the SQL Anywhere Extension Agent to find out which variables are supported.

- **saSrvMetaData.saSrvStatMetaDataTable** Lists the database server statistics (variables under sa.saServer.saSrvStat).
- **saSrvMetaData.saSrvpropMetaDataTable** Lists the database server properties (variables under sa.saServer.saSrv.Prop).
- **saDbMetaData.saDbStatMetaDataTable** Lists the database statistics (variables under sa.saDb.saDbStat).
- **saDbMetaData.saDbpropMetaDataTable** Lists the database properties (variables under sa.saDb.saDbProp).
- **saDbMetaData.saDbOptMetaDataTable** Lists the database options (variables under sa.saDb.saDbOpt).

For more information about the information stored in the SQL Anywhere MIB metadata tables, see "saMetaData tables" on page 1079.

The RDBMS MIB

The RDBMS MIB is a generic and vendor-independent MIB (RFC 1697) for relational database management system products. The RDBMS MIB uses **virtual tables** to return information on the servers and databases. The base OID is **1.3.6.1.2.1.39**, and there are 9 virtual tables in this MIB. The SQL Anywhere SNMP Extension Agent supports eight of these virtual tables.

For more information about the tables contained in the RDBMS MIB, see "RDBMS MIB reference" on page 1107.

The SQL Anywhere Extension Agent provides read-only access to all the supported variables in the RDBMS MIB. None of the variables in the RDBMS MIB are writable through the SQL Anywhere Extension Agent.

A virtual table contains a fixed number of attributes and any number for rows. Elements in the table are retrieved using `GET` requests by appending the column number and row number to the OID of the table. A 1 must be appended to the table OID, so the OID looks as follows:

```
table.1.column.rownum
```

By default, the RDBMS MIB is located in *install-dir\snmp\RDBMS-MIB.mib*.

Using the SQL Anywhere SNMP Extension Agent

To use the SQL Anywhere SNMP Extension Agent, you must have SNMP installed on your computer and you must create an *sasnmp.ini* file that contains information about the databases that are monitored by the SQL Anywhere SNMP Extension Agent.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Installing SNMP

Before you can use the SQL Anywhere Extension Agent, you must install SNMP on your computer. By default, SNMP is not installed on Windows.

For information about installing SNMP, see your operating system documentation.

Once you install SNMP on your computer, the following services should be running on your computer: SNMP Service and SNMP Trap Service.

If you installed SNMP before you installed SQL Anywhere, you need to stop and restart the SNMP service so it can detect the SQL Anywhere SNMP Extension Agent. If you installed SQL Anywhere and then installed SNMP, the SNMP service detects the SQL Anywhere SNMP Extension Agent automatically.

To restart the SNMP service (command line)

1. Run the following command:

   ```
   net stop snmp
   ```

   This stops the SNMP service.

2. Run the following command:

   ```
   net start snmp
   ```

   This starts the SNMP service.

Configuring the SQL Anywhere SNMP Extension Agent

The SQL Anywhere Extension Agent can monitor one or more databases. The databases to be monitored are stored in the *sasnmp.ini* file with the following format:

```
[SAAgent]
TrapPollTime=time-in-seconds

[DBn]
ConnStr=connection-string
UtilDbPwd=utility-database-password
CacheTime=time-in-seconds
DBSpaceCacheTime=time-in-seconds
Trapt=trap-information
Disabled=1 or 0
```

By default, your SQL Anywhere installation places the *sasnmp.ini* file in the *install-dir\bin32* directory.

The SAAgent section

The SAAgent section of the *sasnmp.ini* file contains information about the SQL Anywhere Extension Agent. If the TrapPollTime field is not required, you can omit the entire section.

TrapPollTime This value specifies the poll frequency for dynamic traps if they are specified. The SQL Anywhere SNMP Extension Agent polls the values every 5 seconds by default. Setting this value to 0 disables dynamic traps. This field is optional.

The DBn section

Each **DB*n*** section of the *sasnmp.ini* file describes a database, how to connect to it, and any dynamic traps that exist for the database. The fields in this section are case sensitive.

The value for *n* is a number that identifies the database. The numbers must start with 1, and numbers cannot be skipped. For example, if the *sasnmp.ini* file contained entries for [DB1], [DB2], and [DB4], the [DB4] entry would be ignored because the file is missing the entry for [DB3].

ConnStr The connection string used to connect to the database. You must supply enough information to be able to connect to the database. This field is required.

- If you want to use an ODBC data source to connect to the database, it must be a *system* data source, not a *user* data source.
- If you want to use an integrated login, you must map to the SYSTEM account because the SNMP Agent runs as a service. However, this means that anything that runs as a service can then connect to the database without a password. Alternatively, you can change the account that the service runs under and then create an integrated login for that account.
- The string `ASTART=NO;IDLE=0;CON=SNMP;ASTOP=NO` is prepended to the connection string. This string does the following:
  - prevents the SQL Anywhere SNMP Extension Agent from trying to start a database server automatically
  - disables idle timeout since it is likely that the SQL Anywhere SNMP Extension Agent will sit idle for some time
  - names the connection so it can be identified
  - prevents the database from being shut down when the SQL Anywhere SNMP Extension Agent disconnects

  If you specify any of these values in the connection string in the *sasnmp.ini* file, the values in the *sasnmp.ini* file will override the default settings.

UtilDbPwd When setting `sa.agent.saStarted` to start a database, the SQL Anywhere SNMP Extension Agent attempts to connect to the database with the DBF parameter, which tells the database server where to find the database file. However, if the permission required to start the database is DBA (the default for the network server, which can also be set using the -gd dba option for both the personal and network servers), then the server will not allow the connection.

To start a database on such a server, the SQL Anywhere SNMP Extension Agent must connect as a user with DBA authority to a database already running on the same server. This can be done by connecting to the utility database. If you specify the utility database password (specified by the -su server option) in the *sasnmp.ini* file, then to start a database, the SQL Anywhere Extension Agent connects to the utility

database on the same server, executes the START DATABASE statement, and then disconnects. This field is optional.

CacheTime When data is retrieved from the database, it can be cached inside the SQL Anywhere SNMP Extension Agent, so that subsequent retrievals of the same type of data (for example, server properties or database statistics) do not require communication with the database. While caching the data means that you can obtain the data more quickly on subsequent retrievals, the data may be out of date. The CacheTime field can be used to change the cache time, or disable the cache by setting the value to 0. By default, the cache time is 0 seconds. When the CacheTime parameter is set to 0, the data retrieved is always up-to-date because data is retrieved from the database for every request. This field is optional.

DBSpaceCacheTime The rdbmsDbLimitedResourceTable in the RDBMS MIB contains information about dbspaces. When this information is retrieved from the database, it can also be cached inside the SQL Anywhere Extension Agent. The default cache time for dbspace information is 600 seconds (10 minutes). This field can be used to change the cache time (or disable the cache by setting the value to 0). This field is optional. See "rdbmsDbLimitedResourceTable" on page 1109.

Trap *t* Creates a dynamic trap. The value *t* must be a positive integer starting at 1. Skipping numbers is not allowed. This field is optional. See "Creating dynamic traps" on page 1077.

Disabled If set to 1, this database entry is skipped by the SQL Anywhere SNMP Extension Agent. This is useful for temporarily removing one database from the list of databases managed by the SQL Anywhere SNMP Extension Agent, without renumbering the rest. This field is optional.

Once you edit this file, you must restart the SNMP service or reset the SQL Anywhere SNMP Extension Agent so that the new settings are used by the Agent.

Restarting SNMP

To restart the SNMP service (command line)

1. Run the following command:

   ```
   net stop snmp
   ```

   This stops the SNMP service.

2. Execute the following command:

   ```
   net start snmp
   ```

   This starts the SNMP service.

To restart the SQL Anywhere SNMP Extension Agent

- Using your SNMP management tool, change the value of the saAgent.saRestart property, `1.3.6.1.4.1.897.2.3.6`, to **1**.

You can obfuscate the contents of the *sasnmp.ini* file with simple encryption using the File Hiding utility (dbfhide). See "Hiding the contents of .ini files" on page 397.

Sample sasnmp.ini file

The following is a sample *sasnmp.ini* file for the SQL Anywhere SNMP Extension Agent.

```
[SAAgent]
[DB1]
ConnStr=UID=DBA;PWD=sql;DBN=sales;DBF=sales.db
Trap1=1.1.5 > 50000
UtilDbPwd=test
[DB2]
ConnStr=UID=DBA;PWD=sql;DBN=field;DBF=field.db
UtilDbPwd=test
Disabled=1
[DB3]
ConnStr=UID=DBA;PWD=sql;Host=host2;DBN=hq;DBF=hq.db
UtilDbPwd=test
```

Because there are no parameters specified in the SAAgent section, the SQL Anywhere SNMP Extension Agent will poll values every 5 seconds.

The SQL Anywhere SNMP Extension Agent is monitoring 3 different databases running on two different servers. Database 3 is running on a different computer, so the Host connection parameter is required. A trap is specified for DB1, which fires when the number of bytes sent by the database server is greater than 50000.

Obtaining values using the SQL Anywhere SNMP Extension Agent

Using the SQL Anywhere SNMP Extension Agent, you can retrieve the values of all the following:

- Database server properties. See "SQL Anywhere MIB server properties" on page 1085.
- Database server statistics. See "SQL Anywhere MIB server statistics" on page 1082.
- Database options. See "SQL Anywhere MIB database options" on page 1098.
- Database properties. See "SQL Anywhere MIB database properties" on page 1094.
- Database statistics. See "SQL Anywhere MIB database statistics" on page 1091.

The way you retrieve these values depends on your SNMP management software.

Examples

The table below provides a description and sample value that could be returned for the following OIDs.

OID	Explanation	Sample value
1.3.6.1.4.1.897.2.1.1.1.1	Server statistic ActiveReq on database 1	1
1.3.6.1.4.1.897.2.2.1.4.1	Database statistic CacheRead on database 1	11397

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Explanation	Sample value
1.3.6.1.4.1.897.2.3.1	Agent version	12.0.0(2459)
1.3.6.1.4.1.897.2.3.2.1	Connection string for database 1	UID=DBA;PWD=sql; Host=host1; DBN=sales; DBF=sales.db

Setting values using the SQL Anywhere SNMP Extension Agent

The SQL Anywhere SNMP Extension Agent responds to SNMP get, get-next, and set queries.

You can set any database option, some server properties, and one database property using the SQL Anywhere SNMP agent.

When setting database options, the SQL Anywhere SNMP agent executes the statement:

```
SET OPTION PUBLIC.option-name = 'value'
```

When setting database and server properties, the sa_server_option system procedure is used.

The way you set these values depends on your SNMP management software.

For more information about the options and properties that can be set with the SQL Anywhere SNMP Extension Agent, see “SQL Anywhere MIB reference” on page 1078.

See also

- “SET OPTION statement” [*SQL Anywhere Server - SQL Reference*]
- “Database options” on page 486
- “sa_server_option system procedure” [*SQL Anywhere Server - SQL Reference*]

Executing stored procedures using the SQL Anywhere SNMP Extension Agent

The SQL Anywhere MIB includes an OID that allows you to execute a stored procedure using the SQL Anywhere SNMP Extension Agent. To execute the stored procedure, the user that the SQL Anywhere SNMP Extension Agent uses to connect must have one of the following:

- execute permission on the procedure
- be the owner of the procedure
- have DBA authority

Any result sets or return values generated by the procedure are ignored.

To execute a stored procedure using the SQL Anywhere SNMP Extension Agent, set the value of `saAgent.saProc` (OID 1.3.6.1.4.1.897.2.3.5.*db*, where *db* is the database number in the *sasnmp.ini* file) to a string that is the name of a stored procedure. Optionally, you can supply arguments to the procedure; if no arguments are supplied, parentheses are appended to the procedure name.

For example, setting the value of `saAgent.saProc` to the string "`pchin.updatesales( 'param1', 2)`" calls the updatesales stored procedure owned by user pchin.

The way you set the value of this OID to the procedure name depends on your SNMP management software. See "The SQL Anywhere MIB" on page 1068.

Using traps

A **trap** is an OID that is sent by an SNMP agent when a particular event occurs. Traps are initiated by the SNMP agent and can be detected by SNMP management software, which can then either deal with the event directly or query the SNMP agent for more information.

To receive traps, you must configure the SNMP service. The SNMP service will receive the trap information and then forward it on somewhere; however, by default, this is nowhere, so any trap listeners you have running will not detect anything. The following steps show how to configure your SNMP Service to send traps to your computer.

To configure the SNMP service

1. Right-click **My Computer** and choose **Manage**.
2. In the left pane, double-click **Services And Applications**.
3. In the left pane, double-click **Services**.
4. Locate SNMP Service in the list of services in the right pane, right-click it and choose **Properties**.
5. Click the **Traps** tab.
6. Click **Add**.
7. In the **SNMP Service Configuration** window, type **localhost** in the text box and then click **Add**.
8. Click **OK**.

SQL Anywhere SNMP Extension Agent traps

The SQL Anywhere SNMP Extension Agent sends a trap whenever a connection is dropped by the database server. The OID of this trap is `1.3.6.1.2.1.39.2.1`.

If you are using database mirroring, and the SQL Anywhere SNMP Extension Agent connection to the database server drops, every 30 seconds the SQL Anywhere SNMP Extension Agent attempts to reconnect to the database server. When the agent reconnects, if it finds that it is connected to a different

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

database server (as determined by the ServerName property), then it sends a trap with the OID 1.3.6.1.4.1.897.2.6.3, and the database ID from the *sasnmp.ini* file. In this case, the SQL Anywhere SNMP Extension Agent was connected to the primary database server, which went down, and now the mirror server is acting as the primary server. See "Introduction to database mirroring" on page 945.

The only other traps sent by the SQL Anywhere SNMP Extension Agent are dynamic traps. See "Creating dynamic traps" on page 1077.

Creating dynamic traps

A **dynamic trap** is a trap that is sent by the SQL Anywhere Extension Agent when a simple expression involving the value of a particular property, statistic, or option is true. Dynamic traps are created in the *sasnmp.ini* file. The format of the trap information in the *sasnmp.ini* file entry is as follows:

```
Trap<trapnum>=[1.3.6.1.4.1.897.2.]<oid>[.<dbnum>] <op> <value>
```

trapnum is the dynamic trap number. It must start at 1 and be sequential.

oid is the OID of the property, statistic, or option. OIDs in either the SQL Anywhere MIB or the RDBMS MIB are supported. If the OID given is an invalid SQL Anywhere or RDBMS OID, the SQL Anywhere MIB prefix (1.3.6.1.4.1.897.2.) is prepended.

For information about the OIDs in the SQL Anywhere MIB, see "SQL Anywhere MIB reference" on page 1078.

For information about the OIDs in the RDBMS MIB, see "RDBMS MIB reference" on page 1107.

Note
You can only use OIDs corresponding to database server or database properties, statistics, or options in dynamic traps.

dbnum is the database number. This field is optional, but if specified, it must match the database number of the [DB*n*] section of the *sasnmp.ini* file.

op must have one of the following values:

- = or == (equality)
- **!**=, <>, or >< (inequality)
- <= or =< (less than or equal)
- >= or => (greater than or equal)
- < (less than)
- > (greater than)

Note
Only equality or inequality is supported for string values.

value is the value to use in the expression. String values may be enclosed in single or double quotes; these quotes are not included in the value. If you want the beginning or closing quotation marks to be included in the string, you must double them. Note that single quotes occurring within the string should not be doubled.

When setting dynamic traps, use k, m, g, or t to specify units of kilobytes, megabytes, gigabytes, or terabytes. For example, you can set a dynamic trap to fire if the current cache size exceeds 200 MB by using:

```
Trap1=1.3.6.1.4.1.897.2.1.1.11.1 > 200M
```

You can specify as many Trap fields as you want in the *sasnmp.ini* file. The OID used for the trap is `1.3.6.1.4.1.897.2.4.1`, and the data sent with the trap includes the following:

- the trap number (starts at 1 for the first dynamic trap sent by the SQL Anywhere SNMP agent)
- the database index
- the database name trap index (from the *sasnmp.ini* file)
- the variable name
- the variable value (this is the current value of the variable, not necessarily the threshold value)

Dynamic trap behavior

Once a dynamic trap is triggered, the trap is not sent again until the condition that caused it to be triggered changes to FALSE and then back to TRUE again.

For example, if you have a dynamic trap set using 1.1.11.1 >= 51200K, then the trap is triggered when the server's cache size reaches 50 MB (= 51200 KB) and the dynamic trap is disabled, so no more traps are sent. The only way the trap is re-enabled is if the cache size later drops below 50 MB. You would then be notified if the cache size grew to 50 MB again.

Trap examples

Trap information	Description
Trap1=1.1.5 > 10000	Trap sent when the number of bytes sent from the server is greater than 10000.
Trap2=1.3.6.1.2.1.39.1.4.1.4.14.1 >= 10485760	Trap sent if the size of the transaction log file is larger than 10 MB.

SQL Anywhere MIB reference

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

The list of object identifiers (OIDs) that an SNMP agent supports, including their names, types, and other information are stored in a file called a Management Information Base (MIB). The following sections list the statistics, properties, and options that can be retrieved and set using the SQL Anywhere SNMP Extension Agent.

See also

- "Understanding SNMP" on page 1067

Agent

The Agent table lists information about the SQL Anywhere SNMP Extension Agent.

Writable properties are marked with an asterisk (*). The value *n* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.4.1.897.2.3.1	String	saVersion	Agent version
1.3.6.1.4.1.897.2.3.2.*n*	String	saDBConnStr	Connection string
1.3.6.1.4.1.897.2.3.3.*n*	Integer32	saConnected*	1 if the agent is connected, 0 otherwise
1.3.6.1.4.1.897.2.3.4.*n*	Integer32	saStarted*	1 if the database is started, 0 otherwise
1.3.6.1.4.1.897.2.3.5.*n*	String	saProc*	" "
1.3.6.1.4.1.897.2.3.6	String	saRestart*	0

saMetaData tables

The following metadata tables are included in the SQL Anywhere MIB:

- saSrvMetaData.saSrvStatMetaDataTable
- saSrvMetaData.saSrvPropMetaDataTable
- saSrvMetaData.saDbStatMetaDataTable
- saSrvMetaData.saDbPropMetaDataTable
- saSrvMetaData.saDbOptMetaDataTable

saSrvMetaData.saSrvStatMetaDataTable

This table contains metadata about the database server statistics.

The value *db* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.4.1.897.2.4.1.1.1.1.*db*	Integer32	saSrvStatIndex	*db*
1.3.6.1.4.1.897.2.4.1.1.1.2.*db*	Integer32	saSrvStatObjType	1[1]
1.3.6.1.4.1.897.2.4.1.1.1.3.*db*	Integer32	saSrvStatType	1[2]
1.3.6.1.4.1.897.2.4.1.1.1.4.*db*	OID	saSrvStatOID	OID of SQL Anywhere MIB entry[3]
1.3.6.1.4.1.897.2.4.1.1.1.5.*db*	String	saSrvStatName	Statistic name

[1] Values: 1=Server, 2=Database

[2] Values: 1=Statistic, 2=Property, 3=Option

[3] The OID returned does not include the database number. You must append the database number to the OID before it can be used in a query.

saSrvMetaData.saSrvPropMetaDataTable

This table contains metadata about the database server properties.

The value *db* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.4.1.897.2.4.1.2.1.1.*db*	Integer32	saSrvPropIndex	*db*
1.3.6.1.4.1.897.2.4.1.2.1.2.*db*	Integer32	saSrvPropObjType	1[1]
1.3.6.1.4.1.897.2.4.1.2.1.3.*db*	Integer32	saSrvPropType	2[2]
1.3.6.1.4.1.897.2.4.1.2.1.4.*db*	OID	saSrvPropOID	OID of SQL Anywhere MIB entry[3]
1.3.6.1.4.1.897.2.4.1.2.1.5.*db*	String	saSrvPropName	Property name

[1] Values: 1=Server, 2=Database

[2] Values: 1=Statistic, 2=Property, 3=Option

[3] The OID returned does not include the database number. You must append the database number to the OID before it can be used in a query.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

saDbMetaData.saDbStatMetaDataTable

This table contains metadata about the database statistics.

The value *db* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.4.1.897.2.4.2.1.1.1.*db*	Integer32	saDbStatIndex	*db*
1.3.6.1.4.1.897.2.4.2.1.1.2.*db*	Integer32	saDbStatObjType	2[1]
1.3.6.1.4.1.897.2.4.2.1.1.3.*db*	Integer32	saDbStatType	1[2]
1.3.6.1.4.1.897.2.4.2.1.1.4.*db*	OID	saDbStatOID	OID of SQL Anywhere MIB entry[3]
1.3.6.1.4.1.897.2.4.2.1.1.5.*db*	String	saDbStatName	Statistic name

[1] Values: 1=Server, 2=Database

[2] Values: 1=Statistic, 2=Property, 3=Option

[3] The OID returned does not include the database number. You must append the database number to the OID before it can be used in a query.

saDbMetaData.saDbPropMetaDataTable

This table contains metadata about the database properties.

The value *db* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.4.1.897.2.4.2.2.1.1.*db*	Integer32	saDbPropIndex	*db*
1.3.6.1.4.1.897.2.4.2.2.1.2.*db*	Integer32	saDbPropObjType	2[1]
1.3.6.1.4.1.897.2.4.2.2.1.3.*db*	Integer32	saDbPropType	2[2]
1.3.6.1.4.1.897.2.4.2.2.1.4.*db*	OID	saDbPropOID	OID of SQL Anywhere MIB entry[3]
1.3.6.1.4.1.897.2.4.2.2.1.5.*db*	String	saDbPropName	Property name

[1] Values: 1=Server, 2=Database

[2] Values: 1=Statistic, 2=Property, 3=Option

[3] The OID returned does not include the database number. You must append the database number to the OID before it can be used in a query.

saDbMetaData.saDbOptMetaDataTable

This table contains metadata about the database options.

The value *db* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.4.1.897.2.4.2.1.1.1.*db*	Integer32	saDbOptIndex	*db*
1.3.6.1.4.1.897.2.4.2.1.1.2.*db*	Integer32	saDbOptObjType	2[1]
1.3.6.1.4.1.897.2.4.2.1.1.3.*db*	Integer32	saDbOptType	3[2]
1.3.6.1.4.1.897.2.4.2.1.1.4.*db*	OID	saDbOptOID	OID of SQL Anywhere MIB entry[3]
1.3.6.1.4.1.897.2.4.2.1.1.5.*db*	String	saDbOptName	Option name

[1] Values: 1=Server, 2=Database

[2] Values: 1=Statistic, 2=Property, 3=Option

[3] The OID returned does not include the database number. You must append the database number to the OID before it can be used in a query.

SQL Anywhere MIB server statistics

This table lists the OIDs and names of the database server statistics that can be retrieved using the SQL Anywhere SNMP Extension Agent.

The value *n* is the database number in the *sasnmp.ini* file.

For more information about the database server statistics, see "Database server properties" on page 644 and "Database properties" on page 659.

OID	Type	Name	Statistic
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatActiveReq	ActiveReq
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatAvailIO	AvailIO
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatBytesReceived	BytesReceived

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Statistic
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatBytesReceivedUncomp	BytesReceivedUncomp
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatBytesSent	BytesSent
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatBytesSentUncomp	BytesSentUncomp
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatCacheHits	CacheHits
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatCachePinned	CachePinned
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatCacheRead	CacheRead
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatCacheReplacements	CacheReplacements
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatCurrentCacheSize	CurrentCacheSize
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatDiskRead	DiskRead
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatFreeBuffers	FreeBuffers
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatInternal	Internal
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatUniqueClientAddresses	UniqueClientAddresses
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatLockedHeapPages	LockedHeapPages
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatMainHeapBytes	MainHeapBytes
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatMainHeapPages	MainHeapPages
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatMapPhysicalMemoryEng	MapPhysicalMemoryEng
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatMaxCacheSize	MaxCacheSize
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatMinCacheSize	MinCacheSize
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatMultiPacketsReceived	MultiPacketsReceived
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatMultiPacketsSent	MultiPacketsSent
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatPacketsReceived	PacketsReceived
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatPacketsReceivedUncomp	PacketsReceivedUncomp
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatPacketsSent	PacketsSent
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatPacketsSentUncomp	PacketsSentUncomp

OID	Type	Name	Statistic
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatPeakCacheSize	PeakCacheSize
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatRemoteputWait	RemoteputWait
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatReq	Req
1.3.6.1.4.1.897.2.1.1.*n*	Counter64	srvStatSendFail	SendFail
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatTotalBuffers	TotalBuffers
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatUnschReq	UnschReq
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatInternal	Internal
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatCacheFile	CacheFile
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatCacheFileDirty	CacheFileDirty
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatCacheAllocated	CacheAllocated
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatCachePanics	CachePanics
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatCacheFree	CacheFree
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatCacheScavenges	CacheScavenges
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatCacheScavengeVisited	CacheScavengeVisited
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatLockedCursorPages	LockedCursorPages
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatQueryHeapPages	QueryHeapPages
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatCarverHeapPages	CarverHeapPages
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatHeapsRelocatable	HeapsRelocatable
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatHeapsLocked	HeapsLocked
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatHeapsQuery	HeapsQuery
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatHeapsCarver	HeapsCarver
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatMultiPageAllocs	MultiPageAllocs
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatRequestsReceived	RequestsReceived
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatExchangeTasks	ExchangeTasks

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Statistic
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatClientStmtCacheHits	ClientStmtCacheHits
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatClientStmtCacheMisses	ClientStmtCacheMisses
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatQueryMemActiveCurr	QueryMemActiveCurr
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatQueryMemActiveEst	QueryMemActiveEst
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatQueryMemGrantWaiting	QueryMemGrantWaiting
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatQueryMemGrantRequested	QueryMemGrantRequested
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatQueryMemGrantWaited	QueryMemGrantWaited
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatQueryMemGrantFailed	QueryMemGrantFailed
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatQueryMemGrantGranted	QueryMemGrantGranted
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatQueryMemExtraAvail	QueryMemExtraAvail
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatThreadDeadlocksAvoided	ThreadDeadlocksAvoided
1.3.6.1.4.1.897.2.1.1.*n*	Integer32	srvStatThreadDeadlocksReported	ThreadDeadlocksReported

SQL Anywhere MIB server properties

The following table lists OIDs and names of the database server properties that can be retrieved using the SQL Anywhere SNMP Extension Agent.

Writable properties are marked with an asterisk (*). The value *n* is the database number in the *sasnmp.ini* file.

For more information about the database server properties, see "Database server properties" on page 644.

OID	Type	Name	Property
1.3.6.1.4.1.897.2.1.2.1.*n*	String	srvPropInternal	Internal
1.3.6.1.4.1.897.2.1.2.2.*n*	String	srvPropCharSet	CharSet
1.3.6.1.4.1.897.2.1.2.3.*n*	String	srvPropCommandLine	CommandLine
1.3.6.1.4.1.897.2.1.2.4.*n*	String	srvPropCompactPlatformVer	CompactPlatformVer
1.3.6.1.4.1.897.2.1.2.5.*n*	String	srvPropCompanyName	CompanyName

OID	Type	Name	Property
1.3.6.1.4.1.897.2.1.2.6.*n*	String	srvPropConnsDisabled*	ConnsDisabled
1.3.6.1.4.1.897.2.1.2.7.*n*	String	srvPropConsoleLogFile	ConsoleLogFile
1.3.6.1.4.1.897.2.1.2.8.*n*	String	srvPropDefaultCollation	DefaultCollation
1.3.6.1.4.1.897.2.1.2.9.*n*	String	srvPropIdleTimeout	IdleTimeout
1.3.6.1.4.1.897.2.1.2.10.*n*	String	srvPropIsIQ	IsIQ
1.3.6.1.4.1.897.2.1.2.11.*n*	String	srvPropInternal	Internal
1.3.6.1.4.1.897.2.1.2.12.*n*	String	srvPropIsNetworkServer	IsNetworkServer
1.3.6.1.4.1.897.2.1.2.13.*n*	String	srvPropIsRuntimeServer	IsRuntimeServer
1.3.6.1.4.1.897.2.1.2.14.*n*	String	srvPropInternal	Internal
1.3.6.1.4.1.897.2.1.2.15.*n*	String	srvPropLanguage	Language
1.3.6.1.4.1.897.2.1.2.16.*n*	String	srvPropLegalCopyright	LegalCopyright
1.3.6.1.4.1.897.2.1.2.17.*n*	String	srvPropLegalTrademarks	LegalTrademarks
1.3.6.1.4.1.897.2.1.2.18.*n*	String	srvPropLicenseCount	LicenseCount
1.3.6.1.4.1.897.2.1.2.19.*n*	String	srvPropLicensedCompany	LicensedCompany
1.3.6.1.4.1.897.2.1.2.20.*n*	String	srvPropLicensedUser	LicensedUser
1.3.6.1.4.1.897.2.1.2.21.*n*	String	srvPropLicenseType	LicenseType
1.3.6.1.4.1.897.2.1.2.22.*n*	String	srvPropLivenessTimeout*	LivenessTimeout
1.3.6.1.4.1.897.2.1.2.23.*n*	String	srvPropMachineName	MachineName
1.3.6.1.4.1.897.2.1.2.24.*n*	String	srvPropMaxMessage	MaxMessage
1.3.6.1.4.1.897.2.1.2.25.*n*	String	srvPropMessageWindowSize	MessageWindowSize
1.3.6.1.4.1.897.2.1.2.26.*n*	String	srvPropName	Name
1.3.6.1.4.1.897.2.1.2.27.*n*	String	srvPropNativeProcessorArchitec-ture	NativeProcessorArchitecture
1.3.6.1.4.1.897.2.1.2.28.*n*	String	srvPropNumPhysicalProcessors	NumPhysicalProcessors
1.3.6.1.4.1.897.2.1.2.29.*n*	String	srvPropInternal	Internal

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Property
1.3.6.1.4.1.897.2.1.2.30.*n*	String	srvPropOmniIdentifier	OmniIdentifier
1.3.6.1.4.1.897.2.1.2.31.*n*	String	srvPropPageSize	PageSize
1.3.6.1.4.1.897.2.1.2.32.*n*	String	srvPropPlatform	Platform
1.3.6.1.4.1.897.2.1.2.33.*n*	String	srvPropPlatformVer	PlatformVer
1.3.6.1.4.1.897.2.1.2.34.*n*	String	srvPropProcessCPU	ProcessCPU
1.3.6.1.4.1.897.2.1.2.35.*n*	String	srvPropProcessCPUSystem	ProcessCPUSystem
1.3.6.1.4.1.897.2.1.2.36.*n*	String	srvPropProcessCPUUser	ProcessCPUUser
1.3.6.1.4.1.897.2.1.2.37.*n*	String	srvPropProcessorArchitecture	ProcessorArchitecture
1.3.6.1.4.1.897.2.1.2.38.*n*	String	srvPropProductName	ProductName
1.3.6.1.4.1.897.2.1.2.39.*n*	String	srvPropProductVersion	ProductVersion
1.3.6.1.4.1.897.2.1.2.40.*n*	String	srvPropQuittingTime*	QuittingTime
1.3.6.1.4.1.897.2.1.2.41.*n*	String	srvPropRememberLastStatement*	RememberLastStatement
1.3.6.1.4.1.897.2.1.2.42.*n*	String	srvPropRequestFilterConn	RequestFilterConn
1.3.6.1.4.1.897.2.1.2.43.*n*	String	srvPropRequestFilterDB	RequestFilterDB
1.3.6.1.4.1.897.2.1.2.44.*n*	String	srvPropRequestLogFile*	RequestLogFile
1.3.6.1.4.1.897.2.1.2.45.*n*	String	srvPropRequestLogging*	RequestLogging
1.3.6.1.4.1.897.2.1.2.46.*n*	String	srvPropRequestLogMaxSize	RequestLogMaxSize
1.3.6.1.4.1.897.2.1.2.47.*n*	String	srvPropStartTime	StartTime
1.3.6.1.4.1.897.2.1.2.48.*n*	String	srvPropTempDir	TempDir
1.3.6.1.4.1.897.2.1.2.49.*n*	String	srvPropMultiProgrammingLevel	MultiProgrammingLevel
1.3.6.1.4.1.897.2.1.2.50.*n*	String	srvPropTimeZoneAdjustment	TimeZoneAdjustment
1.3.6.1.4.1.897.2.1.2.51.*n*	String	srvPropHttpPorts	HttpPorts
1.3.6.1.4.1.897.2.1.2.52.*n*	String	srvPropHttpsPorts	HttpsPorts
1.3.6.1.4.1.897.2.1.2.53.*n*	String	srvPropProfileFilterConn	ProfileFilterConn

OID	Type	Name	Property
1.3.6.1.4.1.897.2.1.2.54.*n*	String	srvPropProfileFilterUser	ProfileFilterUser
1.3.6.1.4.1.897.2.1.2.55.*n*	String	srvPropRequestLogNumFiles	RequestLogNumFiles
1.3.6.1.4.1.897.2.1.2.56.*n*	String	srvPropIsFipsAvailable	IsFipsAvailable
1.3.6.1.4.1.897.2.1.2.57.*n*	String	srvPropFipsMode	FipsMode
1.3.6.1.4.1.897.2.1.2.58.*n*	String	srvPropStartDBPermission	StartDBPermission
1.3.6.1.4.1.897.2.1.2.59.*n*	String	srvPropServerName	ServerName
1.3.6.1.4.1.897.2.1.2.60.*n*	String	srvPropRememberLastPlan	RememberLastPlan
1.3.6.1.4.1.897.2.1.2.61.*n*	String	srvPropInternal	Internal
1.3.6.1.4.1.897.2.1.2.62.*n*	String	srvPropInternal	Internal
1.3.6.1.4.1.897.2.1.2.63.*n*	String	srvPropRequestTiming	RequestTiming
1.3.6.1.4.1.897.2.1.2.64.*n*	String	srvPropCacheSizingStatistics	CacheSizingStatistics
1.3.6.1.4.1.897.2.1.2.65.*n*	String	srvPropConsoleLogMaxSize	ConsoleLogMaxSize
1.3.6.1.4.1.897.2.1.2.66.*n*	String	srvPropDebuggingInformation	DebuggingInformation
1.3.6.1.4.1.897.2.1.2.67.*n*	String	srvPropMessage	Message
1.3.6.1.4.1.897.2.1.2.68.*n*	String	srvPropMessageText	MessageText
1.3.6.1.4.1.897.2.1.2.69.*n*	String	srvPropMessageTime	MessageTime
1.3.6.1.4.1.897.2.1.2.70.*n*	String	srvPropIsRsaAvailable	IsRsaAvailable
1.3.6.1.4.1.897.2.1.2.71.*n*	String	srvPropIsEccAvailable	IsEccAvailable
1.3.6.1.4.1.897.2.1.2.72.*n*	String	srvPropMaxConnections	MaxConnections
1.3.6.1.4.1.897.2.1.2.73.*n*	String	srvPropNumLogicalProcessors	NumLogicalProcessors
1.3.6.1.4.1.897.2.1.2.74.*n*	String	srvPropNumLogicalProcessor-sUsed	NumLogicalProcessorsUsed
1.3.6.1.4.1.897.2.1.2.75.*n*	String	srvPropNumPhysicalProcessor-sUsed	NumPhysicalProcessorsUsed
1.3.6.1.4.1.897.2.1.2.76.*n*	String	srvPropDefaultNcharCollation	DefaultNcharCollation

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Property
1.3.6.1.4.1.897.2.1.2.77.*n*	String	srvPropCollectStatistics	CollectStatistics
1.3.6.1.4.1.897.2.1.2.78.*n*	String	srvPropFirstOption	FirstOption
1.3.6.1.4.1.897.2.1.2.79.*n*	String	srvPropLastOption	LastOption
1.3.6.1.4.1.897.2.1.2.80.*n*	String	srvPropLastConnectionProperty	LastConnectionProperty
1.3.6.1.4.1.897.2.1.2.81.*n*	String	srvPropLastDatabaseProperty	LastDatabaseProperty
1.3.6.1.4.1.897.2.1.2.82.*n*	String	srvPropLastServerProperty	LastServerProperty
1.3.6.1.4.1.897.2.1.2.83.*n*	String	srvPropWebClientLogging	WebClientLogging
1.3.6.1.4.1.897.2.1.2.84.*n*	String	srvPropWebClientLogFile	WebClientLogFile
1.3.6.1.4.1.897.2.1.2.85.*n*	String	srvPropHttpNumConnections	HttpNumConnections
1.3.6.1.4.1.897.2.1.2.86.*n*	String	srvPropHttpsNumConnections	HttpsNumConnections
1.3.6.1.4.1.897.2.1.2.87.*n*	String	srvPropInternal	Internal
1.3.6.1.4.1.897.2.1.2.88.*n*	String	srvPropHttpNumActiveReq	HttpNumActiveReq
1.3.6.1.4.1.897.2.1.2.89.*n*	String	srvPropHttpsNumActiveReq	HttpsNumActiveReq
1.3.6.1.4.1.897.2.1.2.90.*n*	String	srvPropHttpNumSessions	HttpNumSessions
1.3.6.1.4.1.897.2.1.2.91.*n*	String	srvPropQueryMemPages	QueryMemPages
1.3.6.1.4.1.897.2.1.2.92.*n*	String	srvPropQueryMemGrantBase	QueryMemGrantBase
1.3.6.1.4.1.897.2.1.2.93.*n*	String	srvPropQueryMemGrantBaseMI	QueryMemGrantBaseMI
1.3.6.1.4.1.897.2.1.2.94.*n*	String	srvPropQueryMemGrantExtra	QueryMemGrantExtra
1.3.6.1.4.1.897.2.1.2.95.*n*	String	srvPropQueryMemActiveMax	QueryMemActiveMax
1.3.6.1.4.1.897.2.1.2.96.*n*	String	srvPropQueryMemPercentOfC-ache	QueryMemPercentOfCache
1.3.6.1.4.1.897.2.1.2.97.*n*	String	srvPropMessageCategoryLimit	MessageCategoryLimit
1.3.6.1.4.1.897.2.1.2.98.*n*	String	srvPropInternal	Internal
1.3.6.1.4.1.897.2.1.2.99.*n*	String	srvPropInternal	Internal

OID	Type	Name	Property
1.3.6.1.4.1.897.2.1.2.100.*n*	String	srvPropIsService	IsService
1.3.6.1.4.1.897.2.1.2.101.*n*	String	srvPropTcpIpAddresses	TcpIpAddresses
1.3.6.1.4.1.897.2.1.2.102.*n*	String	srvPropHttpAddresses	HttpAddresses
1.3.6.1.4.1.897.2.1.2.103.*n*	String	srvPropHttpsAddresses	HttpsAddresses
1.3.6.1.4.1.897.2.1.2.104.*n*	String	srvPropInternal	Internal
1.3.6.1.4.1.897.2.1.2.105.*n*	String	srvPropRemoteCapability	RemoteCapability
1.3.6.1.4.1.897.2.1.2.106.*n*	String	srvPropMaxRemoteCapability	MaxRemoteCapability
1.3.6.1.4.1.897.2.1.2.107.*n*	String	srvPropEventTypeName	EventTypeName
1.3.6.1.4.1.897.2.1.2.108.*n*	String	srvPropEventTypeDesc	EventTypeDesc
1.3.6.1.4.1.897.2.1.2.109.*n*	String	srvPropMaxEventType	MaxEventType
1.3.6.1.4.1.897.2.1.2.110.*n*	String	srvPropIsPortableDevice	IsPortableDevice
1.3.6.1.4.1.897.2.1.2.111.*n*	String	srvPropIPAddressMonitorPeriod	IPAddressMonitorPeriod
1.3.6.1.4.1.897.2.1.2.112.*n*	String	srvPropServerEdition	ServerEdition
1.3.6.1.4.1.897.2.1.2.113.*n*	String	srvPropObjectType	ObjectType
1.3.6.1.4.1.897.2.1.2.114.*n*	String	srvPropCurrentMultiProgrammingLevel	CurrentMultiProgrammingLevel
1.3.6.1.4.1.897.2.1.2.115.*n*	String	srvPropMinMultiProgrammingLevel	MinMultiProgrammingLevel

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Property
1.3.6.1.4.1.897.2.1.2.116.*n*	String	srvPropMaxMultiProgrammingLevel	MaxMultiProgrammingLevel
1.3.6.1.4.1.897.2.1.2.117.*n*	String	srvPropAutoMultiProgrammingLevel	AutoMultiProgrammingLevel
1.3.6.1.4.1.897.2.1.2.118.*n*	String	srvPropAutoMultiProgrammingLevelStatistics	AutoMultiProgrammingLevelStatistics

SQL Anywhere MIB database statistics

The following table lists the OIDs and names the database statistics that can be retrieved using the SQL Anywhere SNMP Extension Agent.

The value *n* is the database number in the *sasnmp.ini* file.

For more information about the database statistics, see "Database server properties" on page 644 and "Database properties" on page 659.

OID	Type	Name	Statistic
1.3.6.1.4.1.897.2.2.1.*n*	Counter64	dbStatCacheHits	CacheHits
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCacheReadIndInt	CacheReadIndInt
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCacheReadIndLeaf	CacheReadIndLeaf
1.3.6.1.4.1.897.2.2.1.*n*	Counter64	dbStatCacheRead	CacheRead
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCacheReadTable	CacheReadTable
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatChkpt	Chkpt
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatChkptFlush	ChkptFlush
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatChkptPage	ChkptPage
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCheckpointUrgency	CheckpointUrgency
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatInternal	Internal
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatInternal	Internal
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCheckpointLogCommitToDisk	CheckpointLogCommitToDisk

OID	Type	Name	Statistic
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCheckpointLogPagesInUse	CheckpointLogPagesInUse
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCheckpointLogPagesRelocated	CheckpointLogPagesRelocated
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCheckpointLogSavePreimage	CheckpointLogSavePreimage
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCheckpointLogSize	CheckpointLogSize
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCheckpointLogWrites	CheckpointLogWrites
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCheckpointLogPagesWritten	CheckpointLogPagesWritten
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCommitFile	CommitFile
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatConnCount	ConnCount
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCurrIO	CurrIO
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCurrRead	CurrRead
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCurrWrite	CurrWrite
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatDiskReadIndInt	DiskReadIndInt
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatDiskReadIndLeaf	DiskReadIndLeaf
1.3.6.1.4.1.897.2.2.1.*n*	Counter64	dbStatDiskRead	DiskRead
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatDiskReadTable	DiskReadTable
1.3.6.1.4.1.897.2.2.1.*n*	Counter64	dbStatDiskWrite	DiskWrite
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatExtendDB	ExtendDB
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatExtendTempWrite	ExtendTempWrite
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatFullCompare	FullCompare
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatGetData	GetData
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatIdleCheck	IdleCheck
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatIdleChkpt	IdleChkpt

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Statistic
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatIdleChkTime	IdleChkTime
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatIdleWrite	IdleWrite
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatIndAdd	IndAdd
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatIndLookup	IndLookup
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatIOToRecover	IOToRecover
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatInternal	Internal
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatInternal	Internal
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatLockTablePages	LockTablePages
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatInternal	Internal
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatMaxIO	MaxIO
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatMaxRead	MaxRead
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatMaxWrite	MaxWrite
1.3.6.1.4.1.897.2.2.1.*n*	Counter64	dbStatPageRelocations	PageRelocations
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatProcedurePages	ProcedurePages
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatQueryCachePages	QueryCachePages
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatQueryLowMemoryStrat-egy	QueryLowMemoryStrategy
1.3.6.1.4.1.897.2.2.1.*n*	Counter64	dbStatQueryRowsMaterialized	QueryRowsMaterialized
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatRecoveryUrgency	RecoveryUrgency
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatLogFreeCommit	LogFreeCommit
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatLogWrite	LogWrite
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatRelocatableHeapPages	RelocatableHeapPages
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatRollbackLogPages	RollbackLogPages
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatTempTablePages	TempTablePages
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatTriggerPages	TriggerPages

OID	Type	Name	Statistic
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatViewPages	ViewPages
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatVersionStorePages	VersionStorePages
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatSnapshotCount	SnapshotCount
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatLockCount	LockCount
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatCacheReadWorkTable	CacheReadWorkTable
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatDiskReadWorkTable	DiskReadWorkTable
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatPrepares	Prepares
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatConnPoolCachedCount	ConnPoolCachedCount
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatConnPoolHits	ConnPoolHits
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatConnPoolMisses	ConnPoolMisses
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatHttpConnPoolCached-Count	HttpConnPoolCachedCount
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatHttpConnPoolHits	HttpConnPoolHits
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatHttpConnPoolMisses	HttpConnPoolMisses
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatHttpConnPoolSteals	HttpConnPoolSteals
1.3.6.1.4.1.897.2.2.1.*n*	Integer32	dbStatMirrorServerWaits	MirrorServerWaits

SQL Anywhere MIB database properties

The following table lists the OIDs and names of the database properties that can be retrieved using the SQL Anywhere SNMP Extension Agent.

Writable properties are marked with an asterisk (*). The value *n* is the database number in the *sasnmp.ini* file.

For more information about the database properties, see "Database properties" on page 659.

OID	Type	Name	Property
1.3.6.1.4.1.897.2.2.2.1.*n*	String	dbPropAlias	Alias

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Property
1.3.6.1.4.1.897.2.2.2.2.*n*	String	dbPropAuditingTypes	AuditingTypes
1.3.6.1.4.1.897.2.2.2.3.*n*	String	dbPropBlankPadding	BlankPadding
1.3.6.1.4.1.897.2.2.2.4.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.5.*n*	String	dbPropCapabilities	Capabilities
1.3.6.1.4.1.897.2.2.2.6.*n*	String	dbPropCaseSensitive	CaseSensitive
1.3.6.1.4.1.897.2.2.2.7.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.8.*n*	String	dbPropCharSet	CharSet
1.3.6.1.4.1.897.2.2.2.9.*n*	String	dbPropChecksum	Checksum
1.3.6.1.4.1.897.2.2.2.10.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.11.*n*	String	dbPropCollation	Collation
1.3.6.1.4.1.897.2.2.2.12.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.13.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.14.*n*	String	dbPropCurrentRedoPos	CurrentRedoPos
1.3.6.1.4.1.897.2.2.2.15.*n*	String	dbPropDBFileFragments	DBFileFragments
1.3.6.1.4.1.897.2.2.2.16.*n*	String	dbPropDriveType	DriveType
1.3.6.1.4.1.897.2.2.2.17.*n*	String	dbPropEncryption	Encryption
1.3.6.1.4.1.897.2.2.2.18.*n*	String	dbPropFile	File
1.3.6.1.4.1.897.2.2.2.19.*n*	String	dbPropFileSize	FileSize
1.3.6.1.4.1.897.2.2.2.20.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.21.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.22.*n*	String	dbPropFreePages	FreePages
1.3.6.1.4.1.897.2.2.2.23.*n*	String	dbPropGlobalDBID	GlobalDBID
1.3.6.1.4.1.897.2.2.2.24.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.25.*n*	String	dbPropInternal	Internal

OID	Type	Name	Property
1.3.6.1.4.1.897.2.2.2.26.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.27.*n*	String	dbPropIQStore	IQStore
1.3.6.1.4.1.897.2.2.2.28.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.29.*n*	String	dbPropLanguage	Language
1.3.6.1.4.1.897.2.2.2.30.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.31.*n*	String	dbPropLogFileFragments	LogFileFragments
1.3.6.1.4.1.897.2.2.2.32.*n*	String	dbPropLogMirrorName	LogMirrorName
1.3.6.1.4.1.897.2.2.2.33.*n*	String	dbPropLogName	LogName
1.3.6.1.4.1.897.2.2.2.34.*n*	String	dbPropLTMGeneration	LTMGeneration
1.3.6.1.4.1.897.2.2.2.35.*n*	String	dbPropLTMTrunc	LTMTrunc
1.3.6.1.4.1.897.2.2.2.36.*n*	String	dbPropMultiByteCharSet	MultiByteCharSet
1.3.6.1.4.1.897.2.2.2.37.*n*	String	dbPropName	Name
1.3.6.1.4.1.897.2.2.2.38.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.39.*n*	String	dbPropPageSize	PageSize
1.3.6.1.4.1.897.2.2.2.40.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.41.*n*	String	dbPropProcedureProfiling*	ProcedureProfiling
1.3.6.1.4.1.897.2.2.2.42.*n*	String	dbPropReadOnly	ReadOnly
1.3.6.1.4.1.897.2.2.2.43.*n*	String	dbPropRemoteTrunc	RemoteTrunc
1.3.6.1.4.1.897.2.2.2.44.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.45.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.46.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.47.*n*	String	dbPropSyncTrunc	SyncTrunc
1.3.6.1.4.1.897.2.2.2.48.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.49.*n*	String	dbPropInternal	Internal

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Property
1.3.6.1.4.1.897.2.2.2.50.*n*	String	dbPropTempFileName	TempFileName
1.3.6.1.4.1.897.2.2.2.51.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.52.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.53.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.54.*n*	String	dbPropNextScheduleTime	NextScheduleTime
1.3.6.1.4.1.897.2.2.2.55.*n*	String	dbPropIdentitySignature	IdentitySignature
1.3.6.1.4.1.897.2.2.2.56.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.57.*n*	String	dbPropSnapshotIsolationState	SnapshotIsolationState
1.3.6.1.4.1.897.2.2.2.58.*n*	String	dbPropConnsDisabled	ConnsDisabled
1.3.6.1.4.1.897.2.2.2.59.*n*	String	dbPropPartnerState	PartnerState
1.3.6.1.4.1.897.2.2.2.60.*n*	String	dbPropArbiterState	ArbiterState
1.3.6.1.4.1.897.2.2.2.61.*n*	String	dbPropMirrorState	MirrorState
1.3.6.1.4.1.897.2.2.2.62.*n*	String	dbPropAlternateServerName	AlternateServerName
1.3.6.1.4.1.897.2.2.2.63.*n*	String	dbPropEncryptionScope	EncryptionScope
1.3.6.1.4.1.897.2.2.2.64.*n*	String	dbPropNcharCharSet	NcharCharSet
1.3.6.1.4.1.897.2.2.2.65.*n*	String	dbPropNcharCollation	NcharCollation
1.3.6.1.4.1.897.2.2.2.66.*n*	String	dbPropAccentSensitive	AccentSensitive
1.3.6.1.4.1.897.2.2.2.67.*n*	String	dbPropSendingTracingTo	SendingTracingTo
1.3.6.1.4.1.897.2.2.2.68.*n*	String	dbPropReceivingTracingFrom	ReceivingTracingFrom
1.3.6.1.4.1.897.2.2.2.69.*n*	String	dbPropIOParallelism	IOParallelism
1.3.6.1.4.1.897.2.2.2.70.*n*	String	dbPropJavaVM	JavaVM
1.3.6.1.4.1.897.2.2.2.71.*n*	String	dbPropDatabaseCleaner	DatabaseCleaner
1.3.6.1.4.1.897.2.2.2.72.*n*	String	dbPropHasCollationTailoring	HasCollationTailoring
1.3.6.1.4.1.897.2.2.2.73.*n*	String	dbPropCatalogCollation	CatalogCollation

OID	Type	Name	Property
1.3.6.1.4.1.897.2.2.2.74.*n*	String	dbPropHasEndianSwapFix	HasEndianSwapFix
1.3.6.1.4.1.897.2.2.2.75.*n*	String	dbPropAlternateMirrorServer-Name	AlternateMirrorServerName
1.3.6.1.4.1.897.2.2.2.76.*n*	String	dbPropOptionWatchList	OptionWatchList
1.3.6.1.4.1.897.2.2.2.77.*n*	String	dbPropOptionWatchAction	OptionWatchAction
1.3.6.1.4.1.897.2.2.2.78.*n*	String	dbPropMirrorMode	MirrorMode
1.3.6.1.4.1.897.2.2.2.79.*n*	String	dbPropHasNCHARLegacyCol-lationFix	HasNCHARLegacyCollation-Fix
1.3.6.1.4.1.897.2.2.2.80.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.81.*n*	String	dbPropAuthenticated	Authenticated
1.3.6.1.4.1.897.2.2.2.82.*n*	String	dbPropSynchronizationSchema-ChangeActive	SynchronizationSchemaChan-geActive
1.3.6.1.4.1.897.2.2.2.83.*n*	String	dbPropLastCheckpointTime	LastCheckpointTime
1.3.6.1.4.1.897.2.2.2.84.*n*	String	dbPropMirrorServerState	MirrorServerState
1.3.6.1.4.1.897.2.2.2.85.*n*	String	dbPropDriveBus	DriveBus
1.3.6.1.4.1.897.2.2.2.86.*n*	String	dbPropDriveModel	DriveModel
1.3.6.1.4.1.897.2.2.2.87.*n*	String	dbPropInternal	Internal
1.3.6.1.4.1.897.2.2.2.88.*n*	String	dbPropMirrorRole	MirrorRole
1.3.6.1.4.1.897.2.2.2.89.*n*	String	dbPropWriteChecksum	WriteChecksum

SQL Anywhere MIB database options

The following table lists the OIDs and names of the database options that can be retrieved using the SQL Anywhere SNMP Extension Agent.

Writable options are marked with an asterisk (*). The value *n* is the database number in the *sasnmp.ini* file.

For more information about the database options, see "Alphabetical list of options" on page 505.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Option
1.3.6.1.4.1.897.2.2.3.1.*n*	String	dbOptAllowNullsBy-Default*	allow_nulls_by_default
1.3.6.1.4.1.897.2.2.3.2.*n*	String	dbOptAnsinull*	ansinull
1.3.6.1.4.1.897.2.2.3.3.*n*	String	dbOptAnsiBlanks*	ansi_blanks
1.3.6.1.4.1.897.2.2.3.4.*n*	String	dbOptAnsiCloseCursor-sOnRollback*	ansi_close_cursors_on_rollback
1.3.6.1.4.1.897.2.2.3.5.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.6.*n*	String	dbOptAnsiPermissions*	ansi_permissions
1.3.6.1.4.1.897.2.2.3.7.*n*	String	dbOptAnsiUpdateCon-straints*	ansi_update_constraints
1.3.6.1.4.1.897.2.2.3.8.*n*	String	dbOptAuditing*	auditing
1.3.6.1.4.1.897.2.2.3.9.*n*	String	dbOptAuditingOptions*	auditing_options
1.3.6.1.4.1.897.2.2.3.10.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.11.*n*	String	dbOptBackgroundPrior-ity*	background_priority
1.3.6.1.4.1.897.2.2.3.12.*n*	String	dbOptBlocking*	blocking
1.3.6.1.4.1.897.2.2.3.13.*n*	Integer32	dbOptBlockingTime-out*	blocking_timeout
1.3.6.1.4.1.897.2.2.3.14.*n*	String	dbOptChained*	chained
1.3.6.1.4.1.897.2.2.3.15.*n*	Integer32	dbOptCheckpointTime*	checkpoint_time
1.3.6.1.4.1.897.2.2.3.16.*n*	Integer32	dbOptCisOption*	cis_option
1.3.6.1.4.1.897.2.2.3.17.*n*	Integer32	dbOptCisRowsetSize*	cis_rowset_size
1.3.6.1.4.1.897.2.2.3.18.*n*	String	dbOptCloseOnEnd-trans*	close_on_endtrans
1.3.6.1.4.1.897.2.2.3.19.*n*	String	dbOptConnectionAu-thentication*	connection_authentication
1.3.6.1.4.1.897.2.2.3.20.*n*	String	dbOptContinueAfter-Raiserror*	continue_after_raiserror

OID	Type	Name	Option
1.3.6.1.4.1.897.2.2.3.21.*n*	String	dbOptConversionError*	conversion_error
1.3.6.1.4.1.897.2.2.3.22.*n*	String	dbOptCooperativeCommits*	cooperative_commits
1.3.6.1.4.1.897.2.2.3.23.*n*	Integer32	dbOptCooperativeCommitTimeout*	cooperative_commit_timeout
1.3.6.1.4.1.897.2.2.3.24.*n*	String	dbOptDatabaseAuthentication*	database_authentication
1.3.6.1.4.1.897.2.2.3.25.*n*	String	dbOptDateFormat*	date_format
1.3.6.1.4.1.897.2.2.3.26.*n*	String	dbOptDateOrder*	date_order
1.3.6.1.4.1.897.2.2.3.27.*n*	String	dbOptDebugMessages*	debug_messages
1.3.6.1.4.1.897.2.2.3.28.*n*	String	dbOptDedicatedTask*	dedicated_task
1.3.6.1.4.1.897.2.2.3.29.*n*	Integer32	dbOptDefaultTimestampIncrement*	default_timestamp_increment
1.3.6.1.4.1.897.2.2.3.30.*n*	String	dbOptDelayedCommits*	delayed_commits
1.3.6.1.4.1.897.2.2.3.31.*n*	Integer32	dbOptDelayedCommitTimeout*	delayed_commit_timeout
1.3.6.1.4.1.897.2.2.3.32.*n*	String	dbOptDivideByZeroError*	divide_by_zero_error
1.3.6.1.4.1.897.2.2.3.33.*n*	String	dbOptEscapeCharacter*	escape_character
1.3.6.1.4.1.897.2.2.3.34.*n*	String	dbOptExcludeOperators*	exclude_operators
1.3.6.1.4.1.897.2.2.3.35.*n*	String	dbOptExtendedJoinSyntax*	extended_join_syntax
1.3.6.1.4.1.897.2.2.3.36.*n*	String	dbOptFireTriggers*	fire_triggers
1.3.6.1.4.1.897.2.2.3.37.*n*	Integer32	dbOptFirstDayOfWeek*	first_day_of_week
1.3.6.1.4.1.897.2.2.3.38.*n*	Integer32	dbOptInternal	Internal

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Option
1.3.6.1.4.1.897.2.2.3.39.*n*	String	dbOptForceViewCreation*	force_view_creation
1.3.6.1.4.1.897.2.2.3.40.*n*	String	dbOptForXmlNullTreatment*	for_xml_null_treatment
1.3.6.1.4.1.897.2.2.3.41.*n*	Integer32	dbOptGlobalDatabaseId*	global_database_id
1.3.6.1.4.1.897.2.2.3.42.*n*	Integer32	dbOptIsolationLevel*	isolation_level
1.3.6.1.4.1.897.2.2.3.43.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.44.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.45.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.46.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.47.*n*	String	dbOptLockRejectedRows*	lock_rejected_rows
1.3.6.1.4.1.897.2.2.3.48.*n*	String	dbOptLoginMode*	login_mode
1.3.6.1.4.1.897.2.2.3.49.*n*	String	dbOptLoginProcedure*	login_procedure
1.3.6.1.4.1.897.2.2.3.50.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.51.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.52.*n*	Integer32	dbOptMaxCursorCount*	max_cursor_count
1.3.6.1.4.1.897.2.2.3.53.*n*	Integer32	dbOptMaxHashSize*	max_hash_size
1.3.6.1.4.1.897.2.2.3.54.*n*	Integer32	dbOptMaxPlansCached*	max_plans_cached
1.3.6.1.4.1.897.2.2.3.55.*n*	Integer32	dbOptMaxRecursiveIterations*	max_recursive_iterations
1.3.6.1.4.1.897.2.2.3.56.*n*	Integer32	dbOptMaxStatementCount*	max_statement_count
1.3.6.1.4.1.897.2.2.3.57.*n*	Integer32	dbOptInternal	Internal

OID	Type	Name	Option
1.3.6.1.4.1.897.2.2.3.58.*n*	Integer32	dbOptMinPasswordLength*	min_password_length
1.3.6.1.4.1.897.2.2.3.59.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.60.*n*	Integer32	dbOptNearestCentury*	nearest_century
1.3.6.1.4.1.897.2.2.3.61.*n*	String	dbOptNonKeywords*	non_keywords
1.3.6.1.4.1.897.2.2.3.62.*n*	String	dbOptOdbcDistinguishCharAndVarchar*	odbc_distinguish_char_and_varchar
1.3.6.1.4.1.897.2.2.3.63.*n*	String	dbOptOnCharsetConversionFailure*	on_charset_conversion_failure
1.3.6.1.4.1.897.2.2.3.64.*n*	String	dbOptOnTsqlError*	on_tsql_error
1.3.6.1.4.1.897.2.2.3.65.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.66.*n*	String	dbOptOptimizationGoal*	optimization_goal
1.3.6.1.4.1.897.2.2.3.67.*n*	Integer32	dbOptOptimizationLevel*	optimization_level
1.3.6.1.4.1.897.2.2.3.68.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.69.*n*	String	dbOptOptimizationWorkload*	optimization_workload
1.3.6.1.4.1.897.2.2.3.70.*n*	Integer32	dbOptPinnedCursorPercentOfCache*	pinned_cursor_percent_of_cache
1.3.6.1.4.1.897.2.2.3.71.*n*	Integer32	dbOptPrecision*	precision
1.3.6.1.4.1.897.2.2.3.72.*n*	String	dbOptPrefetch*	prefetch
1.3.6.1.4.1.897.2.2.3.73.*n*	String	dbOptPreserveSourceFormat*	preserve_source_format
1.3.6.1.4.1.897.2.2.3.74.*n*	String	dbOptPreventArticlePkeyUpdate*	prevent_article_pkey_update
1.3.6.1.4.1.897.2.2.3.75.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.76.*n*	String	dbOptQuotedIdentifier*	quoted_identifier

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Option
1.3.6.1.4.1.897.2.2.3.77.*n*	String	dbOptReadPastDeleted*	read_past_deleted
1.3.6.1.4.1.897.2.2.3.78.*n*	Integer32	dbOptRecoveryTime*	recovery_time
1.3.6.1.4.1.897.2.2.3.79.*n*	String	dbOptReplicateAll*	replicate_all
1.3.6.1.4.1.897.2.2.3.80.*n*	String	dbOptReturnDateTimeAsString*	return_date_time_as_string
1.3.6.1.4.1.897.2.2.3.81.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.82.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.83.*n*	String	dbOptRowCounts*	row_counts
1.3.6.1.4.1.897.2.2.3.84.*n*	Integer32	dbOptScale*	scale
1.3.6.1.4.1.897.2.2.3.85.*n*	String	dbOptSortCollation*	sort_collation
1.3.6.1.4.1.897.2.2.3.86.*n*	String	dbOptSqlFlaggerErrorLevel*	sql_flagger_error_level
1.3.6.1.4.1.897.2.2.3.87.*n*	String	dbOptSqlFlaggerWarningLevel*	sql_flagger_warning_level
1.3.6.1.4.1.897.2.2.3.88.*n*	String	dbOptStringRtruncation*	string_rtruncation
1.3.6.1.4.1.897.2.2.3.89.*n*	String	dbOptSubsumeRowLocks*	subsume_row_locks
1.3.6.1.4.1.897.2.2.3.90.*n*	String	dbOptSuppressTdsDebugging*	suppress_tds_debugging
1.3.6.1.4.1.897.2.2.3.91.*n*	String	dbOptTdsEmptyStringIsNull*	tds_empty_string_is_null
1.3.6.1.4.1.897.2.2.3.92.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.93.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.94.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.95.*n*	String	dbOptTimestampFormat*	timestamp_format

OID	Type	Name	Option
1.3.6.1.4.1.897.2.2.3.96.*n*	String	dbOptTimeFormat*	time_format
1.3.6.1.4.1.897.2.2.3.97.*n*	Integer32	dbOptTimeZoneAdjustment*	time_zone_adjustment
1.3.6.1.4.1.897.2.2.3.98.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.99.*n*	String	dbOptTruncateTimestampValues*	truncate_timestamp_values
1.3.6.1.4.1.897.2.2.3.100.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.101.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.102.*n*	String	dbOptTsqlVariables*	tsql_variables
1.3.6.1.4.1.897.2.2.3.103.*n*	String	dbOptUpdateStatistics*	update_statistics
1.3.6.1.4.1.897.2.2.3.104.*n*	String	dbOptUpgradeDatabaseCapability*	upgrade_database_capability
1.3.6.1.4.1.897.2.2.3.105.*n*	String	dbOptUserEstimates*	user_estimates
1.3.6.1.4.1.897.2.2.3.106.*n*	String	dbOptWaitForCommit*	wait_for_commit
1.3.6.1.4.1.897.2.2.3.107.*n*	String	dbOptTempSpaceLimitCheck*	temp_space_limit_check
1.3.6.1.4.1.897.2.2.3.108.*n*	Integer32	dbOptRemoteIdleTimeout*	remote_idle_timeout
1.3.6.1.4.1.897.2.2.3.109.*n*	String	dbOptAnsiSubstring*	ansi_substring
1.3.6.1.4.1.897.2.2.3.110.*n*	String	dbOptOdbcDescribeBinaryAsVarbinary*	odbc_describe_binary_as_varbinary
1.3.6.1.4.1.897.2.2.3.111.*n*	String	dbOptRollbackOnDeadlock*	rollback_on_deadlock

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Option
1.3.6.1.4.1.897.2.2.3.112.*n*	String	dbOptIntegratedServer-Name*	integrated_server_name
1.3.6.1.4.1.897.2.2.3.113.*n*	String	dbOptLogDeadlocks*	log_deadlocks
1.3.6.1.4.1.897.2.2.3.114.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.115.*n*	String	dbOptWebserviceNa-mespaceHost*	webservice_namespace_host
1.3.6.1.4.1.897.2.2.3.116.*n*	Integer32	dbOptMaxQueryTasks*	max_query_tasks
1.3.6.1.4.1.897.2.2.3.117.*n*	Integer32	dbOptRequestTimeout*	request_timeout
1.3.6.1.4.1.897.2.2.3.118.*n*	String	dbOptSynchronizeMir-rorOnCommit*	synchronize_mirror_on_commit
1.3.6.1.4.1.897.2.2.3.119.*n*	Integer32	dbOptHttpSessionTi-meout*	http_session_timeout
1.3.6.1.4.1.897.2.2.3.120.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.121.*n*	String	dbOptAllowSnapshotI-solation*	allow_snapshot_isolation
1.3.6.1.4.1.897.2.2.3.122.*n*	String	dbOptVerifyPassword-Function*	verify_password_function
1.3.6.1.4.1.897.2.2.3.123.*n*	String	dbOptDefaultDbspace*	default_dbspace
1.3.6.1.4.1.897.2.2.3.124.*n*	String	dbOptCollectStatistic-sOnDmlUpdates*	collect_statistics_on_dml_updates
1.3.6.1.4.1.897.2.2.3.125.*n*	String	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.126.*n*	String	dbOptJavaLocation*	java_location
1.3.6.1.4.1.897.2.2.3.127.*n*	String	dbOptOemString*	oem_string

OID	Type	Name	Option
1.3.6.1.4.1.897.2.2.3.128.*n*	Integer32	dbOptMaxTempSpace*	max_temp_space
1.3.6.1.4.1.897.2.2.3.129.*n*	String	dbOptSecureFeature-Key*	secure_feature_key
1.3.6.1.4.1.897.2.2.3.130.*n*	String	dbOptMaterializedVie-wOptimization*	materialized_view_optimization
1.3.6.1.4.1.897.2.2.3.131.*n*	Integer32	dbOptUpdatableState-mentIsolation*	updatable_statement_isolation
1.3.6.1.4.1.897.2.2.3.132.*n*	String	dbOptTsqlOuterJoins*	tsql_outer_joins
1.3.6.1.4.1.897.2.2.3.133.*n*	String	dbOptPostLoginProce-dure*	post_login_procedure
1.3.6.1.4.1.897.2.2.3.134.*n*	String	dbOptConnAuditing*	conn_auditing
1.3.6.1.4.1.897.2.2.3.135.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.136.*n*	String	dbOptJavaVmOptions*	java_vm_options
1.3.6.1.4.1.897.2.2.3.137.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.138.*n*	Integer32	dbOptMaxClientState-mentsCached*	max_client_statements_cached
1.3.6.1.4.1.897.2.2.3.139.*n*	String	dbOptQueryMemTime-out*	query_mem_timeout
1.3.6.1.4.1.897.2.2.3.140.*n*	String	dbOptAllowRead-ClientFile*	allow_read_client_file
1.3.6.1.4.1.897.2.2.3.141.*n*	String	dbOptAllowWrite-ClientFile*	allow_write_client_file
1.3.6.1.4.1.897.2.2.3.142.*n*	String	dbOptPriority*	priority
1.3.6.1.4.1.897.2.2.3.143.*n*	String	dbOptMaxPriority*	max_priority

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Option
1.3.6.1.4.1.897.2.2.3.144.*n*	String	dbOptProgressMessages*	progress_messages
1.3.6.1.4.1.897.2.2.3.145.*n*	Integer32	dbOptBlockingOthersTimeout*	blocking_others_timeout
1.3.6.1.4.1.897.2.2.3.146.*n*	Integer32	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.147.*n*	String	dbOptTimestampWithTimeZoneFormat*	timestamp_with_time_zone_format
1.3.6.1.4.1.897.2.2.3.148.*n*	Integer32	dbOptHttpConnectionPoolTimeout*	http_connection_pool_timeout
1.3.6.1.4.1.897.2.2.3.149.*n*	Integer32	dbOptHttpConnectionPoolBasesize*	http_connection_pool_basesize
1.3.6.1.4.1.897.2.2.3.150.*n*	String	dbOptStGeometryDescribeType*	st_geometry_describe_type
1.3.6.1.4.1.897.2.2.3.151.*n*	String	dbOptStGeometryAstextFormat*	st_geometry_astext_format
1.3.6.1.4.1.897.2.2.3.152.*n*	String	dbOptStGeometryAsbinaryFormat*	st_geometry_asbinary_format
1.3.6.1.4.1.897.2.2.3.153.*n*	String	dbOptStGeometryAsxmlFormat*	st_geometry_asxml_format
1.3.6.1.4.1.897.2.2.3.154.*n*	String	dbOptReservedKeywords*	reserved_keywords
1.3.6.1.4.1.897.2.2.3.155.*n*	String	dbOptStGeometryOnInvalid*	st_geometry_on_invalid

RDBMS MIB reference

The following sections list the OIDs of the values that can be retrieved using the SQL Anywhere SNMP Extension Agent. By default, the RDBMS MIB is located in *C:\Program Files\SQL Anywhere 12\snmp\RDBMS-MIB.mib*.

rdbmsDbTable

This table lists information about the databases installed on a system.

The value *db* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.2.1.39.1.1.1.1.*db*	Integer	rdbmsDbIndex	*db*
1.3.6.1.2.1.39.1.1.1.2.*db*	OID	rdbmsDbPrivateMibOID	1.3.6.1.4.1.897.2
1.3.6.1.2.1.39.1.1.1.3.*db*	String	rdbmsDbVendorName	PROPERTY('Company-Name')
1.3.6.1.2.1.39.1.1.1.4.*db*	String	rdbmsDbName	DB_PROPERTY('Name')
1.3.6.1.2.1.39.1.1.1.5.*db*	String	rdbmsDbContact	PROPERTY('LicensedUser')

rdbmsDbInfoTable

This table provides additional information about the databases on the system.

The value *db* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.2.1.39.1.2.1.1.*db*	String	rdbmsDbInfoProduct-Name	PROPERTY('ProductName')
1.3.6.1.2.1.39.1.2.1.2.*db*	String	rdbmsDbInfoVersion	PROPERTY('ProductVersion')
1.3.6.1.2.1.39.1.2.1.3.*db*	Integer	rdbmsDbInfoSizeUnits	Calculated based on dbInfoSizeAllocated and dbInfoSizeUsed. • 1=bytes • 2=KB • 3=MB • 4=GB • 5=TB (Each unit is 1024 times the previous.)

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Value returned
1.3.6.1.2.1.39.1.2.1.4.*db*	Integer	rdbmsDbInfoSizeAllocated	DB_PROPERTY('PageSize') * DB_PROPERTY('FileSize')
1.3.6.1.2.1.39.1.2.1.5.*db*	Integer	rdbmsDbInfoSizeUsed	DB_PROPERTY('PageSize') * (DB_PROPERTY('FileSize') - DB_PROPERTY('FreePages'))
1.3.6.1.2.1.39.1.2.1.6.*db*	String	rdbmsDbInfoLastBackup	NULL[1]

[1] This OID is not supported by the SQL Anywhere SNMP Extension Agent.

rdbmsDbParamTable

This table lists the configuration parameters for the databases on the system.

The value *db* is the database number in the *sasnmp.ini* file, while *n* is the index of the option in the `sa.2.3` subtree.

OID	Type	Name	Value returned
1.3.6.1.2.1.39.1.3.1.1.*db*	String	rdbmsDbParamName	Option name
1.3.6.1.2.1.39.1.3.1.2.*db*	Integer	rdbmsDbParamSubIndex	*n*
1.3.6.1.2.1.39.1.3.1.3.*db*	OID	rdbmsDbParamID	OID in SQL Anywhere MIB corresponding to this option
1.3.6.1.2.1.39.1.3.1.4.*db*	String	rdbmsDbParamCurrValue	Option value
1.3.6.1.2.1.39.1.3.1.5.*db*	String	rdbmsDbParamComment	NULL[1]

[1] This OID is not supported by the SQL Anywhere SNMP Extension Agent.

rdbmsDbLimitedResourceTable

This table lists free space information on each dbspace. In this table, *n* represents each dbspace as follows:

- 1-13 are for normal dbspaces (numbered 0-12 in the database)
- 14 is the transaction log file
- 15 is the transaction log mirror file

- 16 is the temporary file

The value *db* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.2.1.39.1.4.1.1.*n.d b*	String	rdbmsDbLimitedResource-Name	Name of dbspace, or Transaction Log, Transaction Log Mirror, or Temporary File.
1.3.6.1.2.1.39.1.4.1.2.*n.d b*	OID	rdbmsDbLimitedResourceID	1.3.6.1.4.1.897.2
1.3.6.1.2.1.39.1.4.1.3.*n.d b*	Integer	rdbmsDbLimitedResource-Limit	Free space available on disk + current file size
1.3.6.1.2.1.39.1.4.1.4.*n.d b*	Integer	rdbmsDbLimitedResource-Current	Current file size
1.3.6.1.2.1.39.1.4.1.5.*n.d b*	Integer	rdbmsDbLimitedResource-Highwater	Current size
1.3.6.1.2.1.39.1.4.1.6.*n.d b*	Integer	rdbmsDbLimitedResource-Failure	0[1]
1.3.6.1.2.1.39.1.4.1.7.*n.d b*	String	rdbmsDbLimitedResource-Description	One of Bytes, KB, MB, GB, or TB.

[1] This OID is not supported by the SQL Anywhere SNMP Extension Agent.

rdbmsSrvTable

This table lists the database servers running or installed on your system.

The value *db* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.2.1.39.1.5.1.1.*db*	OID	rdbmsSrvPrivateMibOID	1.3.6.1.4.1.897.2
1.3.6.1.2.1.39.1.5.1.2.*db*	String	rdbmsSrvVendorName	PROPERTY('CompanyName')
1.3.6.1.2.1.39.1.5.1.3.*db*	String	rdbmsSrvProductName	PROPERTY('ProductName')
1.3.6.1.2.1.39.1.5.1.4.*db*	String	rdbmsSrvContact	PROPERTY('LicensedCompany')

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

rdbmsSrvInfoTable

This table lists additional information about the database servers in your system.

The value *db* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.2.1.39.1.6.1.1.*db*	Integer	rdbmsSrvInfoStartupTime	PROPERTY('StartTime')
1.3.6.1.2.1.39.1.6.1.2.*db*	Integer	rdbmsSrvInfoFinishedTransactions	0[1]
1.3.6.1.2.1.39.1.6.1.3.*db*	Integer	rdbmsSrvInfoDiskReads	PROPERTY('DiskReadEng')
1.3.6.1.2.1.39.1.6.1.4.*db*	Integer	rdbmsSrvInfoLogicalReads	0[1]
1.3.6.1.2.1.39.1.6.1.5.*db*	Integer	rdbmsSrvInfoDiskWrites	PROPERTY('DiskWriteEng')
1.3.6.1.2.1.39.1.6.1.6.*db*	Integer	rdbmsSrvInfoLogicalWrites	0[1]
1.3.6.1.2.1.39.1.6.1.7.*db*	Integer	rdbmsSrvInfoPageReads	0[1]
1.3.6.1.2.1.39.1.6.1.8.*db*	Integer	rdbmsSrvInfoPageDiskOutOfWrites	0[1]
1.3.6.1.2.1.39.1.6.1.9.*db*	Integer	rdbmsSrvInfoSpaces	0[1]
1.3.6.1.2.1.39.1.6.1.10.*db*	Integer	rdbmsSrvInfoHandledRequests	PROPERTY('Req')
1.3.6.1.2.1.39.1.6.1.11.*db*	Integer	rdbmsSrvInfoRequestRecvs	PROPERTY('PacketsReceivedUncomp')
1.3.6.1.2.1.39.1.6.1.12.*db*	Integer	rdbmsSrvInfoRequestSends	PROPERTY('PacketsSentUncomp')
1.3.6.1.2.1.39.1.6.1.13.*db*	Integer	rdbmsSrvInfoHighwaterInboundAssociations	0[1]
1.3.6.1.2.1.39.1.6.1.14.*db*	Integer	rdbmsSrvInfoMaxInboundAssociations	0[1]

[1] This OID is not supported by the SQL Anywhere SNMP Extension Agent.

rdbmsSrvParamTable

This table lists the server options that can be set by the SQL Anywhere SNMP Extension Agent through the SQL Anywhere MIB. *n* is the index, as follows:

n	Server option
1	ConnsDisabled
2	LivenessTimeout (default)
3	QuittingTime
4	RememberLastStatement
5	RequestLogFile
6	RequestLogging

The value *db* is the database number in the *sasnmp.ini* file.

OID	Type	Name	Value returned
1.3.6.1.2.1.39.1.7.1.1.*n.db*	String	rdbmsDbSrvParamName	Name of option *n*
1.3.6.1.2.1.39.1.7.1.2.*n.db*	Integer	rdbmsDbSrvParamSubIndex	*n*
1.3.6.1.2.1.39.1.7.1.3.*n.db*	OID	rdbmsDbSrvParamID	1.3.6.1.4.1.897.2
1.3.6.1.2.1.39.1.7.1.4.*n.db*	String	rdbmsDbSrvParamCurrValue	Current value of option *n*
1.3.6.1.2.1.39.1.7.1.5.*n.db*	String	rdbmsDbSrvParamComment	Full name of option *n*

rdbmsSrvLimitedResourceTable

This table contains information about server configuration parameters.

The value *db* is the database number in the *sasnmp.ini* file, while *n* is the index of the resource as follows:

n	Name	Resource	Resource limit
1	Connections	PROPERTY('UniqueClientAddresses')	PROPERTY('LicenseCount')
2	Processors	PROPERTY('NumLogicalProcessorsUsed')	PROPERTY('NumLogicalProcessorsUsed')

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

OID	Type	Name	Value returned
1.3.6.1.2.1.39.1.8.1.1.*db*	String	rdbmsSrvLimitedResourceName	Name of resource *n*
1.3.6.1.2.1.39.1.8.1.2.*db*	OID	rdbmsSrvLimitedResourceID	OID in SQL Anywhere MIB corresponding to this option
1.3.6.1.2.1.39.1.8.1.3.*db*	Integer	rdbmsSrvLimitedResourceLimit	Upper limit of resource *n*
1.3.6.1.2.1.39.1.8.1.4.*db*	Integer	rdbmsSrvLimitedResourceCurrent	Current value of resource *n*
1.3.6.1.2.1.39.1.8.1.5.*db*	Integer	rdbmsSrvLimitedResourceHighwater	Current value of resource *n*
1.3.6.1.2.1.39.1.8.1.6.*db*	Integer	rdbmsSrvLimitedResourceFailures	0[1]
1.3.6.1.2.1.39.1.8.1.7.*db*	String	rdbmsSrvLimitedResourceDescription	Name of resource *n*

[1] This OID is not supported by the SQL Anywhere SNMP Extension Agent.

Security

This section describes security features in SQL Anywhere.

Keeping your data secure

Since databases may contain proprietary, confidential, or private information, ensuring that the database and the data in it are designed for security is very important.

SQL Anywhere has several features to assist in building a secure environment for your data:

- **User identification and authentication** These features control who has access to a database. See "Creating new users" on page 450.
- **Discretionary access control features** These features control the actions a user can perform while connected to a database. See "Database permissions and authorities" on page 441.
- **Auditing** This feature helps you maintain a record of actions on the database. See "Auditing database activity" on page 1123.
- **Database server options** These features let you control who can perform administrative operations (for example, loading databases). These options are set when you start the database server. See "Controlling permissions from the command line" on page 48.
- **Views and stored procedures** These features allow you to specify the data a user can access and the operations a user can execute. See "Using views and procedures for extra security" on page 480.
- **Database and table encryption** You can choose to secure your database either with simple encryption, or with strong encryption. Simple encryption is equivalent to obfuscation. Strong encryption renders the database completely inaccessible without an encryption key. See "-ek dbeng12/dbsrv12 database option" on page 256 and "DatabaseKey (DBKEY) connection parameter" on page 281.

  Table encryption features allow you to encrypt individual tables, instead of encrypting the entire database. See "Table encryption" on page 1139.
- **Transport-layer security** You can use transport-layer security to authenticate communications between client applications and the database server. Transport-layer security uses elliptic-curve or RSA encryption technology. See "Transport-layer security" on page 1143.

> **Note**
> If you are concerned about other processes on the computer running the database server being able to access the contents of your client/server communications, it is recommended that you use encryption.

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

- **Secured features** You can disable features for all databases running on a database server.
- **SELinux support** SELinux policies control an application's access to system resources. SQL Anywhere includes a policy that secures it on Red Hat Enterprise Linux 5.

  For information about compiling and installing the SQL Anywhere SELinux policy, see *install-dir/selinux/readme*.

Database administrators are responsible for data security. In this section, unless otherwise noted, you require DBA authority to perform the tasks described.

User IDs and permissions are security-related topics. See "Managing user IDs, authorities, and permissions" on page 441.

Security tips

As database administrator, there are many actions you can take to improve the security of your data. For example, you can:

- **Choose passwords carefully** Do not deploy databases that use the default user ID and password. See "Increasing password security" on page 1118.
- **Restrict DBA authority** You should restrict DBA authority only to users who absolutely require it since it is very powerful. Users with DBA authority can see and do anything in the database.

  You may consider giving users with DBA authority two user IDs: one with DBA authority and one without, so they can connect as a DBA user only when necessary.
- **Use secured database features** The database server -sf option lets you enable and disable features for all databases running on a database server. The features you can disable include the use of external stored procedures, Java, remote data access, and the ability to change the request log settings. See "-sf dbeng12/dbsrv12 server option" on page 219 and "Specifying secured features" on page 1122.
- **Drop external system functions** The following external functions present possible security risks: xp_cmdshell, xp_startmail, xp_startsmtp, xp_sendmail, xp_stopmail, and xp_stopsmtp.

  The xp_cmdshell procedure allows users to execute operating system commands or programs.

  The email commands allow users to have the server send email composed by the user. Malicious users could use either the email or command shell procedures to perform operating-system tasks with

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

authorities other than those they have been given by the operating system. In a security-conscious environment, you should drop these functions.

For information about dropping procedures, see "DROP PROCEDURE statement" [*SQL Anywhere Server - SQL Reference*].

- **Protect your database files** You should protect the database file, log files, and dbspace files from unauthorized access. Do not store them within a shared directory or volume.

- **Protect your database software** You should similarly protect SQL Anywhere software. Only give users access to the applications, DLLs, and other resources they require.

- **Run the database server as a service or a daemon** To prevent unauthorized users from shutting down or gaining access to the database or log files, run the database server as a Windows service. On Unix, running the server as a daemon serves a similar purpose. See "Running the database server outside the current session" on page 57.

- **Set SATMP to a unique directory** To make the database server secure on Unix platforms, set SATMP to a unique directory, and make the directory read, write, and execute protected against all other users. Doing so forces all other connections to use TCP/IP, which is more secure than the shared memory connection.

  The shared memory buffers that are used between the client and server are removed from the directory tree before any actual data is sent between the two sides. This means that another process cannot see any of the communication data because the shared memory buffer/file is hidden, and so a process cannot get a handle to it.

- **Strongly encrypt your database** Strongly encrypting your database makes it completely inaccessible without the key. You cannot open the database, or view the database or transaction log files using any other means.

  For more information, see "-ep dbeng12/dbsrv12 server option" on page 179 and "-ek dbeng12/dbsrv12 database option" on page 256.

Controlling database access

By assigning user IDs and passwords, the database administrator controls who has access to a database. By granting permissions to each user ID, the database administrator controls which tasks each user can perform when connected to the database.

Permission scheme is based on user IDs

When a user logs on to the database, they have access to all database objects that meet any of the following criteria:

- objects the user created
- objects to which the user has received explicit permission
- objects to which a group the user belongs to received explicit permission

The user cannot access any database object that does not meet these criteria. In short, users can access only the objects they own or objects to which they explicitly received access permissions.

For more information, see:

- "Managing user IDs, authorities, and permissions" on page 441
- "CONNECT statement [ESQL] [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "REVOKE statement" [*SQL Anywhere Server - SQL Reference*]

Using integrated logins

Integrated logins allow users to use a single login name and password to log onto both your Windows operating system and onto a database. An external login name is associated with a database user ID. When you attempt an integrated login, you log onto the operating system by giving both a login name and password. The operating system then tells the server who you are, and the server logs you in as the associated database user ID. No additional login name or password are required.

When using integrated logins, leaving the user profile Guest enabled with a blank password can permit unrestricted access to a database that is hosted by the server that accepts integrated logins. Literally any user can log in to the server using any login ID and any password because they are logged in by default to the Guest user profile.

For more information, see:

- "Security concerns: Unrestricted database access" on page 115
- "Using Windows integrated logins" on page 108
- "login_mode option" on page 547

Increasing password security

Passwords are an important part of any database security system. To be secure, passwords must be difficult to guess, and they must not be easily accessible on users' hard drives or other locations. SQL Anywhere passwords are always case sensitive. You can specify a function used for password authentication with the verify_password_function option. See "verify_password_function option" on page 614.

Implement a login policy

Use a login policy to control the frequency of user password changes and to specify the number of login attempts allowed before an account is locked. See "Managing login policies" on page 471, or "CREATE LOGIN POLICY statement" [*SQL Anywhere Server - SQL Reference*].

Change the default user ID and password

The default user ID and password for a newly created database is **DBA** and **sql**. You should change this password before deploying the database.

Implement minimum password lengths

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

By default, passwords can be any length. For greater security, you can enforce a minimum length requirement on all new passwords to disallow short (and therefore easily guessed) passwords. You do this by setting the min_password_length database option to a value greater than zero. The following statement enforces passwords to be at least 8 bytes long.

```
SET OPTION PUBLIC.min_password_length = 8;
```

See "min_password_length option" on page 560.

Implement password expiration

By default, database passwords never expire. You can use a login policy to implement password expiry. See "Managing login policies" on page 471.

Do not include passwords in ODBC data sources

Passwords are the key to accessing databases. They should not be easily available to unauthorized people in a security-conscious environment.

When you create an ODBC data source or a Sybase Central connection profile, you can optionally include a password. Avoid including passwords to ensure that they are not viewed by unauthorized users.

See "Creating ODBC data sources" on page 98.

Encrypt configuration files containing passwords

When you create a configuration file, you can optionally include password information. To protect your passwords, consider hiding the contents of configuration files with simple encryption, using the File Hiding (dbfhide) utility. See "File Hiding utility (dbfhide)" on page 794.

Use password verification

You can use the verify_password_function option to specify a function that implements password rules. See "verify_password_function option" on page 614.

The following example defines a table and function and sets some login policy options. Together they implement advanced password rules that include requiring certain types of characters in the password, disallowing password reuse, and expiring passwords. The function is called by the database server with the verify_password_function option when a user ID is created or a password is changed. The application can call the procedure specified by the post_login_procedure option to report that the password should be changed before it expires.

The code for this sample is also available in the following location: *samples-dir\SQLAnywhere\SQL \verify_password.sql*. (For information about *samples-dir*, see "Samples directory" on page 392.)

```
-- This example defines a function that implements advanced password rules
-- including requiring certain types of characters in the password and
-- disallowing password reuse. The f_verify_pwd function is called by the
-- server using the verify_password_function option when a user ID is
-- created or a password is changed.
--
-- The "root" login profile is configured to expire passwords every 180 days
-- and lock non-DBA accounts after 5 consecutive failed login attempts.
--
-- The application may call the procedure specified by the
```

```
-- post_login_procedure option to report that the password should be changed
-- before it expires.


-- only DBA should have permissions on this table
CREATE TABLE DBA.t_pwd_history(
        pk          INT         DEFAULT AUTOINCREMENT PRIMARY KEY,
        user_name   CHAR(128),          -- the user whose password is set
        pwd_hash    CHAR(32) );         -- hash of password value to detect
                                        -- duplicate passwords

-- called whenever a non-NULL password is set
-- to verify the password conforms to password rules
CREATE FUNCTION DBA.f_verify_pwd( uid     VARCHAR(128),
                                  new_pwd VARCHAR(255) )
RETURNS VARCHAR(255)
BEGIN
    -- a table with one row per character in new_pwd
    DECLARE local temporary table pwd_chars(
            pos INT PRIMARY KEY,    -- index of c in new_pwd
            c   CHAR( 1 CHAR ) );   -- character
    -- new_pwd with non-alpha characters removed
    DECLARE pwd_alpha_only      CHAR(255);
    DECLARE num_lower_chars     INT;

    -- enforce minimum length (can also be done with
    -- min_password_length option)
    IF length( new_pwd ) < 6 THEN
        RETURN 'password must be at least 6 characters long';
    END IF;

    -- break new_pwd into one row per character
    INSERT INTO pwd_chars SELECT row_num, substr( new_pwd, row_num, 1 )
                          FROM dbo.RowGenerator
                          WHERE row_num <= length( new_pwd );

    -- copy of new_pwd containing alpha-only characters
    SELECT list( c, '' ORDER BY pos ) INTO pwd_alpha_only
        FROM pwd_chars WHERE c BETWEEN 'a' AND 'z' OR c BETWEEN 'A' AND 'Z';

    -- number of lower case characters IN new_pwd
    SELECT count(*) INTO num_lower_chars
        FROM pwd_chars WHERE CAST( c AS BINARY ) BETWEEN 'a' AND 'z';

    -- enforce rules based on characters contained in new_pwd
    IF ( SELECT count(*) FROM pwd_chars WHERE c BETWEEN '0' AND '9' )
           < 1 THEN
        RETURN 'password must contain at least one numeric digit';
    ELSEIF length( pwd_alpha_only ) < 2 THEN
        RETURN 'password must contain at least two letters';
    ELSEIF num_lower_chars = 0
           OR length( pwd_alpha_only ) - num_lower_chars = 0 THEN
        RETURN 'password must contain both upper- and lowercase characters';
    END IF;

    -- not the same as any user name
    -- (this could be modified to check against a disallowed words table)
    IF EXISTS( SELECT * FROM SYS.SYSUSER
                   WHERE lower( user_name ) IN ( lower( pwd_alpha_only ),
                                                 lower( new_pwd ) ) ) THEN
        RETURN 'password or only alphabetic characters in password ' ||
               'must not match any user name';
    END IF;
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
    -- not the same as any previous password for this user
    IF EXISTS( SELECT * FROM t_pwd_history
                   WHERE user_name = uid
                     AND pwd_hash = hash( uid || new_pwd, 'md5' ) ) THEN
        RETURN 'previous passwords cannot be reused';
    END IF;

    -- save the new password
    INSERT INTO t_pwd_history( user_name, pwd_hash )
        VALUES( uid, hash( uid || new_pwd, 'md5' ) );

    RETURN( NULL );
END;

ALTER FUNCTION DBA.f_verify_pwd SET HIDDEN;
GRANT EXECUTE ON DBA.f_verify_pwd TO PUBLIC;
SET OPTION PUBLIC.verify_password_function = 'DBA.f_verify_pwd';


-- All passwords expire in 180 days.  Expired passwords can be changed
-- by the user using the NewPassword connection parameter.
ALTER LOGIN POLICY root password_life_time = 180;

-- If an application calls the procedure specified by the
-- post_login_procedure option, then the procedure can be used to warn
-- the user that their password is about to expire.  In particular,
-- Interactive SQL and Sybase Central call the
-- post_login_procedure system procedure.
ALTER LOGIN POLICY root password_grace_time = 30;

-- Five consecutive failed login attempts will result in a non-DBA
-- user ID being locked.
ALTER LOGIN POLICY root max_failed_login_attempts = 5;
```

Controlling the tasks users can perform

You can control the tasks users can perform on database objects (such as creating, modifying, executing, updating, and so on) by granting permissions. You can control the administrative tasks (such as backing up, profiling, and so on) that a user can perform by granting authorities.

You grant permissions and authorities using the GRANT statement. For permissions, you can also delegate permission granting privileges on an object to other users.

The REVOKE statement is the opposite of the GRANT statement—any permission that GRANT has explicitly given, REVOKE can take away. Revoking CONNECT from a user removes the user from the database, including all objects owned by that user.

See also

- “Managing user IDs, authorities, and permissions” on page 441
- “GRANT statement” [*SQL Anywhere Server - SQL Reference*]
- “REVOKE statement” [*SQL Anywhere Server - SQL Reference*]

Designing database objects for security

Views and stored procedures provide alternate ways of tuning the data that users can access and the tasks they can perform.

See:

- "Benefits of procedures and triggers" [*SQL Anywhere Server - SQL Usage*]
- "Using views and procedures for extra security" on page 480

Specifying secured features

To control the database features available to users, you can include the secured features option (-sf) when starting the database server. The secured features option controls the availability of such features as:

- server-side backups
- external stored procedures
- remote data access
- web services

For a complete list of features, see "-sf dbeng12/dbsrv12 server option" on page 219.

You also have the option of including the -sk option when you start the database server. This option specifies a key that can be used to re-enable secured features for a specific connection. You re-enable secured features for a connection by setting the value of the secure_feature_key temporary option to the value specified by -sk when the database server was started.

To modify the features or feature sets that are secured for the connection, specify a key with -sk and set the secure_feature_key temporary option to the key value to use the sa_server_option system procedure. Any changes you make to enable or disable features take effect immediately.

To secure database features

1. Start the database server using the -sf, and optionally -sk, options.

   For example, the following command starts the database server and disables the use of remote data access. However, it includes a key that can be used to re-enable the disabled features for a connection.

   ```
   dbsrv12 -n secure_server -sf remote_data_access -sk ls64uwq15 c:\mydata.db
   ```

2. Connect to the database server.

   For example:

   ```
   dbisql -c "UID=DBA;PWD=sql;Host=myhost;Server=secure_server;DBN=demo"
   ```

3. Set the value of the temporary secure_feature_key option to the value specified by -sk when the database server was started.

   For example:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
SET TEMPORARY OPTION secure_feature_key = 'ls64uwq15';
```

4. Change the secured features for the database server with the sa_server_option system procedure.

   For example:

   ```
   CALL sa_server_option( 'SecureFeatures', '-remote_data_access' );
   ```

See also

- "-sf dbeng12/dbsrv12 server option" on page 219
- "-sk dbeng12/dbsrv12 server option" on page 224
- "secure_feature_key" on page 587
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

Auditing database activity

Each database has an associated transaction log file. The transaction log is used for database recovery. It is a record of transactions executed against a database. See "The transaction log" on page 21.

The transaction log stores all executed data definition statements, and the user ID that executed them. It also stores all updates, deletes, and inserts and which user executed those statements. However, this is insufficient for some auditing purposes. By default, the transaction log does not contain the time of the event, just the order in which events occurred. It also contains neither failed events, nor select statements.

Auditing is a way of keeping track of the activity performed on a database. When you use auditing, additional data is saved in the transaction log, including:

- All login attempts (successful and failed), including the terminal ID.
- Accurate timestamps of all events (to a resolution of milliseconds).
- All permissions checks (successful and failed), including the object on which the permission was checked (if applicable).
- All actions that require DBA authority.

You cannot stop using a transaction log while auditing is enabled for a database. If you want to turn off the transaction log, you must first turn off auditing.

Controlling auditing

The database administrator can turn on auditing to add security-related information to the transaction log. This can be done using Sybase Central or Interactive SQL.

Auditing is off by default. You must have DBA authority to enable and disable auditing.

To control auditing (Sybase Central)

1. Use the **SQL Anywhere 12** plug-in to connect to the database as a user with DBA authority.
2. Right-click the database and choose **Properties**.
3. Click the **Auditing** tab and choose one of the following:
   - **Do Not Collect Audit Information For This Database** No audit information is collected. This option disables auditing by setting the auditing database option to Off. See "auditing option" on page 514.
   - **Collect All Audit Information For This Database** All types of auditing information are collected for the database. This option enables auditing by setting the auditing database option to On. See "auditing option" on page 514.

     The transaction log can grow significantly when this option is selected.
   - **Collect The Following Type(s) Of Audit Information For This Database** Allows you to specify which auditing information to collect. For example, you can choose to collect only DDL changes. See "sa_enable_auditing_type system procedure" [*SQL Anywhere Server - SQL Reference*]. Selecting this option changes the setting of the auditing_options database option. See "auditing_options option" on page 515.
4. Click **OK**.

To control auditing (Interactive SQL)

1. Connect to your database as a user with DBA authority.
2. Execute the following statement to turn on auditing:

   ```
   SET OPTION PUBLIC.auditing = 'On';
   ```

   To specify which types of auditing information you want to enable, use the following system procedure:

   ```
   CALL sa_enable_auditing_type( 'all' );
   ```

   You can control the type of auditing information that is collected by replacing **all** with the types of auditing you want to enable. See "sa_enable_auditing_type system procedure" [*SQL Anywhere Server - SQL Reference*].
3. Execute the following statement to turn off auditing:

   ```
   SET OPTION PUBLIC.auditing = 'Off';
   ```

   To specify which types of auditing information you want to disable, use the following system procedure:

   ```
   CALL sa_disable_auditing_type( 'all' );
   ```

   You can stop collecting specific types of auditing information by replacing **all** with the types of auditing you want to disable. See "sa_disable_auditing_type system procedure" [*SQL Anywhere Server - SQL Reference*].

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Auditing individual connections

Once you have enabled auditing for a database, you can set the temporary conn_auditing database option in the database login procedure to enable connection-specific auditing. You can enable auditing based on information such as the IP address of the client computer or the type of connection.

If you do not set the conn_auditing option in the login procedure, the option is on by default.

The following example shows an excerpt from a login procedure that enables auditing for all connections to the database, except those made by the DBA user:

```
DECLARE usr VARCHAR(128)
SELECT CONNECTION_PROPERTY( 'Userid' ) INTO usr;
IF usr != 'DBA' THEN
   SET TEMPORARY OPTION conn_auditing='On'
ELSE
   SET TEMPORARY OPTION conn_auditing='Off'
END IF;
```

For more information, see "login_procedure option" on page 549 and "conn_auditing option" on page 522.

See also

- "auditing option" on page 514

Retrieving auditing information

You can use Sybase Central or the Log Translation utility (dbtran) to retrieve audit information from the transaction log. Before attempting to retrieve audit information, ensure that you are connected to your database as a user with DBA, Remote, or Backup authority.

To retrieve auditing information (Sybase Central)

1. Select the database.
2. Click the **Auditing** tab.
3. Click **Retrieve Audit Messages**.

   A window appears displaying the dbtran messages. Ignore the warning about chronological ordered output.
4. Click **Close**.

   The auditing information appears on the **Auditing** tab in the right pane.
5. Use the filter options to control which audit information you want to display.

   You can choose to display all audit information, or to show only errors, or only audit messages containing the text you specify.

6. Select an entry in the audit entries table to display details about the entry.

7. To retrieve up-to-date auditing information, press F5, and repeat this procedure.

For more information, see "Auditing example" on page 1127.

Retrieving auditing information using the dbtran utility

You can access the dbtran utility from Sybase Central or from a command prompt. The dbtran utility uses the specified transaction log to produce a SQL script that contains all the transactions, and some information about what user executed each command. By using the -g option, dbtran includes more comments containing the auditing information. The -g option is equivalent to specifying the following options:

- **-d** Display output in chronological order.
- **-t** Include trigger-generated operations in the output.
- **-a** Include rolled back transactions in the output.

For more information about these options, see "Log Translation utility (dbtran)" on page 820.

You can run the dbtran utility against a running database server or against a database log file.

To retrieve auditing information from a running database server

- With the database server running, run the following command:

  ```
  dbtran -g -c connection-string -n SQL-file
  ```

  For example:

  ```
  dbtran -g -c "UID=DBA;PWD=sql" -n demo.sql
  ```

  A readable version of the transaction log is saved to your current directory. In the example, the auditing information is saved to the *demo.sql* file, and the file contains information about the sample database.

  For more information about connection strings, see "Connection parameters" on page 265.

To retrieve auditing information from a transaction log file

1. Shut down the database server to ensure the transaction log file is available.

2. Run the following command:

   ```
   dbtran -g transaction-log SQL-file
   ```

For example:

```
dbtran -g demo.log demo.sql
```

In the example, the auditing information from the transaction log file *demo.log* is placed into the file *demo.sql*.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For more information, see "Log Translation utility (dbtran)" on page 820.

Adding audit comments

You can add comments to the audit trail using the sa_audit_string system stored procedure. It takes a single argument, which is a string of up to 200 bytes. You must have DBA authority to call this procedure.

For example:

```
CALL sa_audit_string( 'Started audit testing here.' );
```

This comment is stored in the transaction log as an audit statement.

Auditing example

This example shows how the auditing feature records attempts to access unauthorized information using either Sybase Central or Interactive SQL.

Sybase Central auditing example

1. Start Sybase Central and connect to the sample database using the SQL Anywhere 12 Demo data source.

   This connects you as a DBA user.

2. Turn on auditing:

   a. Right-click the database and choose **Properties**.

   b. Click the **Auditing** tab.

   c. Click **Collect All Audit Information For This Database**.

   d. Click **Apply**.

   e. Click **OK**.

3. Add a user named Test1 to the sample database, with the password welcome:

   a. Right-click **Users & Groups**, and choose **New** » **User**.

   b. When prompted, name the user **Test1**, and type **welcome** as their password.

   c. Give the user **Profile Authority**.

   d. Click **Finish**.

   e. Disconnect from the sample database.

4. Using Sybase Central, connect to the sample database as Test1 and attempt to access confidential information in the Employees table:

   a. Select **Tables**, and then select the Employees table.

   b. Click the **Data** tab.

      An error message appears: `Permission denied: you do not have permission to select from "Employees"`.

   c. Click **OK**.

   d. Disconnect from the sample database.

5. View the auditing information for this activity:

   a. Using Sybase Central, connect to the sample database as a user with DBA authority.

   b. Select the database, and then click the **Auditing** tab in the right pane.

   c. Click **Retrieve Audit Messages**.

   d. Click **Close**.

      Auditing information appears.

   e. Use the filtering options to locate the error in the auditing information table. You can find the error for BadUser by selecting the **Only Errors** option. Use the date and time information to pinpoint the error. For example, if BadUser tried accessing the Employees table on November 6, 2007 at 10:07:14, the corresponding audit entry resembles the following entry:

      ```
      2007-11-06 10:07:14 | Permission
      ```

6. Restore the sample database to its original state:

   a. Right-click the database, and then choose **Properties**.

   b. On the **Auditing** tab, select **Do Not Collect Audit Information For This Database**.

   c. Click **OK**.

   d. Select **Users & Groups**.

      Right-click Test1, and choose **Delete**.

Interactive SQL auditing example

1. Start Interactive SQL and connect to the sample database using the SQL Anywhere 12 Demo data source.

   This connects you as a DBA user.

2. Turn on auditing using the SET OPTION statement, as follows:

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
SET OPTION PUBLIC.auditing = 'On';
```

3. Add a user, Test1, to the sample database using the CREATE USER statement, as follows:

   ```
   CREATE USER Test1
   IDENTIFIED BY welcome;
   ```

4. Open a new Interactive SQL window, connect to the sample database as BadUser, and attempt to access confidential information in the Employees table using the following SELECT statement:

   ```
   SELECT Surname, Salary
      FROM GROUPO.Employees;
   ```

   You receive an error message: `Permission denied: you do not have permission to select from "Employees"`.

5. Run the following command to view the auditing information for this activity:

   ```
   dbtran -g -c "DSN=SQL Anywhere 12 Demo" -n demo.sql
   ```

6. Restore the sample database to its original state:

   - Use the DROP USER statement to remove the Test1 user from the database:

     ```
     DROP USER Test1;
     ```

   - Turn off auditing using the following SET OPTION statement:

     ```
     SET OPTION PUBLIC.auditing = 'Off';
     ```

Auditing actions outside the database server

Some database utilities act on the database file directly. In a secure environment, only trusted users should have access to the database files.

To provide auditing of actions, under Windows or Unix, any use of dbtran or dblog generates a text file in the same directory as the database file, with the extension *.alg*. For example, for *demo.db*, the file is called *demo.alg*. Records containing the tool name, Windows or Unix user name, and date/time are appended to this file. Records are only added to the *.alg* file if the auditing option is set to On.

See also

- “auditing option” on page 514
- “Log Translation utility (dbtran)” on page 820
- “Transaction Log utility (dblog)” on page 862

Running the database server in a secure fashion

There are several security features you can set either when starting the database server or during server operation, including:

- **Starting and stopping databases** When using a personal database server, by default any user can start an extra database on a running server. By default, network database servers require DBA authority to start another database on a running database server. The -gd option allows you to limit access to this option to users with a certain level of permission in the database to which they are already connected. The allowed values are DBA, all, or none. See "-gd dbeng12/dbsrv12 server option" on page 185.

- **Creating and deleting databases** When running a personal database server, by default any user can use the CREATE DATABASE statement to create a database file. By default, network database servers required DBA authority to create databases. The -gu option allows you to limit access to this option to users with a certain level of permission in the database to which they are connected. The permissible values are DBA, all, none, or utility_db. See "-gu dbeng12/dbsrv12 server option" on page 198.

- **Stopping the server** The dbstop utility stops a database server. It is useful in batch files, or in other cases where stopping the server interactively (by clicking **Shut Down** on the database server messages window) is impractical. By default on personal database servers, any user can run dbstop to shut down a server. On network database servers, the default setting requires DBA authority to stop a database server. The -gk option allows you to limit access to this option to users with a certain level of permission in the database. The permissible values are DBA, all, or none. See "-gk dbeng12/dbsrv12 server option" on page 187.

- **Loading and unloading data** The LOAD TABLE, UNLOAD TABLE, and UNLOAD statements all access the file system on the database server computer. The default setting is all for personal database servers on non-Unix operating systems, and DBA for the network database server and the Unix personal server. If you are running the personal database server, you already have access to the file system and this is not a security issue. If you are running the network database server, unwarranted file system access may be a security issue. The -gl option allows you to control the database permissions required to perform loading and unloading of data. The permissible values are DBA, all, or none. See "-gl dbeng12/dbsrv12 server option" on page 188.

- **Using transport-layer security to encrypt client/server communications** For greater security of network packets, you can use transport-layer security to authenticate communications between client applications and the database server. Transport-layer security uses elliptic-curve or RSA encryption technology. See "Transport-layer security" on page 1143.

- **Disabling database features** The -sf server option specifies a list of features that are disabled for databases running on the database server so they are not available to client applications or stored procedures, triggers, or events defined within the databases. This can be useful when you are starting a database that is not your own that may contain unwanted actions, such as a virus or trojan. See "-sf dbeng12/dbsrv12 server option" on page 219.

Encrypting and decrypting a database

As a database administrator, you can use database encryption to make it more difficult for someone to decipher the data in your database. You can choose to secure your database either with simple or with strong encryption.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Note
If your database is encrypted, compressing it with a tool such as WinZip does not result in a file that is significantly smaller than the original database file.

Simple encryption

Simple encryption is equivalent to obfuscation and makes it more difficult for someone using a disk utility to look at the file to decipher the data in your database. Simple encryption does not require a key to encrypt the database. Simple encryption technology is supported in previous versions of SQL Anywhere.

To use simple encryption

- Create a database using the dbinit -ea simple option.

  The following example creates the database *test.db* using simple encryption:

  ```
  dbinit -ea simple test.db
  ```

See also

- "Initialization utility (dbinit)" on page 799
- "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]

Strong encryption

Strong database encryption technology makes a database inoperable and inaccessible without a key (password). An algorithm encodes the information contained in your database and transaction log files so they cannot be deciphered.

Caution
For strongly encrypted databases, be sure to store a copy of the key in a safe location. If you lose the encryption key there is no way to access the data, even with the assistance of technical support. The database must be discarded and you must create a new database.

Supported strong encryption algorithms

The algorithm used to implement SQL Anywhere strong encryption is AES: a block encryption algorithm chosen as the new Advanced Encryption Standard (AES) for block ciphers by the National Institute of Standards and Technology (NIST).

You can also specify a separate FIPS-approved AES algorithm for strong encryption using the AES_FIPS (128-bit) or AES256_FIPS (256-bit) type. When the database server is started with the -fips option, you can run databases encrypted with AES, AES256, AES_FIPS, or AES256_FIPS strong encryption, but not databases encrypted with simple encryption. Unencrypted databases can also be started on the server when -fips is specified. See "-fips dbeng12/dbsrv12 server option" on page 182.

The SQL Anywhere security option must be installed on any computer used to run a database encrypted with AES_FIPS or AES256_FIPS.

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

Note
FIPS is not available on all platforms. For a list of supported platforms, see http://www.sybase.com/detail?id=1061806.

Controlling strong encryption settings for your database

In SQL Anywhere, the database administrator has control over four aspects of strong encryption, including: strong encryption status, the encryption key, protection of the encryption key, and the encryption algorithm.

Although you cannot simply turn strong encryption on or off in an existing database, you can choose from three options when it comes to implementing strong encryption. You can either create a database from scratch with strong encryption, you can rebuild an existing database and change the encryption status at that time, or you can use the CREATE ENCRYPTED DATABASE statement on an existing database.

You can rebuild the database to unload all the data and schema of an existing database. This creates a new database (at which point you can change a variety of settings including strong encryption status), and reloads the data into the new database. You need to know the key to unload a strongly encrypted database.

See also

- "Reload a database" [*SQL Anywhere Server - SQL Usage*]
- "Initialization utility (dbinit)" on page 799
- "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE ENCRYPTED DATABASE statement" [*SQL Anywhere Server - SQL Reference*]

Creating an encrypted database

To create an encrypted database, you can use the following:

- The Initialization utility (dbinit) in combination with various options to enable strong encryption.

  The dbinit utility -ep and -ek options create a database with strong encryption, allowing you to specify the encryption key in a prompt box or on the command line. The dbinit -ea option sets the encryption algorithm to AES or AES256 (or to AES_FIPS or AES256_FIPS for the FIPS-approved algorithm). See "Initialization utility (dbinit)" on page 799.

- The Sybase Central **Create Database Wizard** to create a strongly encrypted database. See "Create a database (Sybase Central)" on page 9.

- The Unload Database utility (dbunload) with options to create a new database with strong encryption. The -an option creates a new database. To specify strong encryption and the encryption key in a

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

prompt box or on the command line use the -ep or -ek option. The -ea option sets the encryption algorithm to AES or AES256 (or to AES_FIPS or AES256_FIPS for the FIPS-approved algorithm). See "Unload utility (dbunload)" on page 864.

- You can also use the Sybase Central **Unload Database Wizard** to create a strongly encrypted database. See "Export data with the Unload Database Wizard" [*SQL Anywhere Server - SQL Usage*].
- The following SQL statements:
  - "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
  - "CREATE ENCRYPTED DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
  - "CREATE ENCRYPTED FILE statement" [*SQL Anywhere Server - SQL Reference*]

To create an encrypted database (SQL)

1. Connect to an existing database from Interactive SQL.
2. Execute a CREATE DATABASE statement that includes the ENCRYPTION clause and the KEY and ALGORITHM options.

   For example, the following statement creates a database file named *myencrypteddb.db* in the *c:* directory using FIPS-approved 128-bit AES encryption.

   ```
   CREATE DATABASE 'c:\\myencrypteddb.db'
   TRANSACTION LOG ON
   ENCRYPTED ON
     KEY '0kZ2o52AK#'
     ALGORITHM 'AES_FIPS';
   ```

To create an encrypted database (command line)

1. Use the dbinit utility to create a database. You must include -ek or -ep to specify the encryption key at the command prompt or a window, respectively.

   The following command creates a strongly encrypted database and specifies the encryption key and algorithm.

   ```
   dbinit -ek "0kZ2o56AK#" -ea AES_FIPS "myencrypteddb.db"
   ```

2. Run the following command to start the database:

   ```
   dbeng12 myencrypteddb.db -ek "0kZ2o56AK#"
   ```

To create an encrypted database using an existing database (SQL)

1. Connect to an existing database (other than the one you are copying) from Interactive SQL.
2. Encrypt the database using the CREATE ENCRYPTED DATABASE statement.

   The following statement takes the database file *demo.db*, and creates an AES-encrypted copy of it named *encryptedDemo.db*.

```
CREATE ENCRYPTED DATABASE 'encryptedDemo.db'
FROM 'demo.db'
KEY 'abc'
ALGORITHM 'AES';
```

When you execute a CREATE ENCRYPTED DATABASE statement, you are not actually encrypting (overwriting) the file; you are creating a copy of the file in encrypted form. If there are transaction logs, transaction log mirrors, or dbspaces associated with the database, encrypted copies of those files are made as well. See "CREATE ENCRYPTED DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

Encrypting a database for technical support

If you have a database that requires recovery and you want to encrypt it to send it to technical support, you must use the CREATE ENCRYPTED FILE statement. Any database-related files such as the transaction log and transaction log mirrors, and dbspace files, must also be encrypted using this statement. See "CREATE ENCRYPTED FILE statement" [*SQL Anywhere Server - SQL Reference*].

Comparison of CREATE ENCRYPTED DATABASE and CREATE ENCRYPTED FILE statements

You should use the CREATE ENCRYPTED DATABASE statement when you have an existing database that you want to encrypt. Use CREATE ENCRYPTED FILE statement only in the case where you have a database you want to encrypt that requires recovery.

Both statements require you to have DBA authority, and you cannot be connected to the database you are encrypting when you execute the statement.

The CREATE ENCRYPTED FILE and CREATE ENCRYPTED DATABASE statements differ from each other as follows:

- The CREATE ENCRYPTED FILE statement must be executed against each of the database-related files independently (transaction log, transaction log mirror, dbspaces, if any), whereas the CREATE ENCRYPTED DATABASE statement automatically encrypts all the database-related files.
- The CREATE ENCRYPTED DATABASE statement cannot be used on a database requiring recovery; the CREATE ENCRYPTED FILE statement can.
- The CREATE ENCRYPTED DATABASE statement cannot be used inside procedures, triggers, or batches. The CREATE ENCRYPTED FILE statement can.
- The CREATE ENCRYPTED DATABASE statement supports the SIMPLE encryption algorithm, but the CREATE ENCRYPTED FILE statement does not.

See also

For more information about encryption keys, see "DatabaseKey (DBKEY) connection parameter" on page 281.

On Windows Mobile, the AES_FIPS and AES256_FIPS algorithms are only supported with ARM processors.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Note
FIPS is not available on all platforms. For a list of supported platforms, see http://www.sybase.com/detail?id=1061806.

Decrypting a database

You can decrypt a database using the CREATE DECRYPTED DATABASE statement. As with the CREATE ENCRYPTED DATABASE statement, you are creating a copy of the file (in this case, in decrypted form), and not actually overwriting the original database file.

To decrypt a database (SQL)

1. Connect to an existing database from Interactive SQL.
2. Decrypt the database using the CREATE DECRYPTED DATABASE statement.

   The first statement creates an AES256-encrypted copy of the *demo.db* database called *demoEncrypted.db*,. The second statement creates a decrypted copy of *demoEncrypted.db* called *demoDecrypted.db*.

   ```
   CREATE ENCRYPTED DATABASE 'demoEncrypted.db'
      FROM 'demo.db'
      KEY 'Sd8f6654*Mnn'
      ALGORITHM 'AES256';
   CREATE DECRYPTED DATABASE 'demoDecrypted.db'
      FROM 'demoEncrypted.db'
      KEY 'Sd8f6654*Mnn';
   ```

   If there are transaction logs, transaction log mirrors, or dbspaces associated with the database, decrypted copies of those files are made as well. See "CREATE DECRYPTED DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

Decrypting a database for technical support

If you have a database that requires recovery and you want to decrypt it to send it to technical support, you must use the CREATE DECRYPTED FILE statement. Any database-related files such as transaction logs and transaction log mirrors, and dbspace files, must also be decrypted using this statement. See "CREATE DECRYPTED FILE statement" [*SQL Anywhere Server - SQL Reference*].

Working with encryption keys

It is best to choose an encryption key value that cannot be easily guessed. The key can be of arbitrary length, but generally the longer the key, the better because a shorter key is easier to guess than a longer one. As well, including a combination of numbers, letters, and special characters decreases the chances of someone guessing the key.

Encryption keys are always case sensitive, and they cannot contain leading or trailing spaces or semicolons.

You must supply this key each time you want to start the database. Lost or forgotten keys result in completely inaccessible databases.

You can choose whether the encryption key is entered at the command prompt (the default) or into a prompt box. Choosing to enter the key in a prompt box provides an extra measure of security because the key is never visible in plain sight. Clients are required to specify the key each time they start the database. If the database administrator starts the database, clients never need to have access to the key. See "-ep dbeng12/dbsrv12 server option" on page 179.

Caution
For strongly encrypted databases, be sure to store a copy of the key in a safe location. If you lose the encryption key there is no way to access the data, even with the assistance of technical support. The database must be discarded and you must create a new database.

You can change the encryption key for an encrypted database, or for a database for which table encryption has been enabled, using the CREATE ENCRYPTED DATABASE statement. As with encrypting the database, you are not overwriting the existing file, you are creating a copy of the file, encrypted with the new key.

To change the encryption key for a database

- Change the encryption key for an encrypted database using the CREATE ENCRYPTED DATABASE statement.

  The following example takes the database file *myOldDatabase.db*, encrypted with key abc, and creates a copy of it called *myNewDatabase.db*, encrypting it with the key abc123. Any other database-related files (transaction log, transaction log mirrors, dbspace files) are also created using the new encryption key. See "CREATE ENCRYPTED DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

```
CREATE ENCRYPTED DATABASE myNewDatabase.db
FROM myOldDatabase.db
KEY 'abc123'
OLD KEY 'abc'
ALGORITHM 'AES';
```

Performance issues

Performance of SQL Anywhere is slower when the database is encrypted. The performance impact depends on how often pages are read from or written to disk, and can be minimized by ensuring that the server is using an adequate cache size.

You can increase the starting size of the cache with the -c option when you start the server. For operating systems that support dynamic resizing of the cache, the cache size that is used may be restricted by the amount of memory that is available; to increase the cache size, increase the available memory.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "Use the cache to improve performance" [*SQL Anywhere Server - SQL Usage*]
- "-c dbeng12/dbsrv12 server option" on page 159

Encrypting portions of a database

If you only want to encrypt portions of your database, you can choose to encrypt columns or tables.

Column encryption can be performed on any column in any table at any time. Table encryption requires that the database have table encryption enabled. Table encryption is enabled at database creation (initialization) time.

Column encryption

If you want to encrypt columns in your database, you can do so with the ENCRYPT function. The ENCRYPT function uses the same AES strong encryption algorithm that is used for database encryption to encrypt values that are passed to it.

The key for the ENCRYPT function is case sensitive, even in case-insensitive databases. As with most passwords, it is best to choose a key value that cannot be easily guessed. It is recommended that you choose a value for your key that is at least 16 characters long, contains a mix of upper and lowercase, and includes numbers, letters and special characters. You must specify this key each time you want to decrypt the data.

Caution
For strongly encrypted databases, be sure to store a copy of the key in a safe location. If you lose the encryption key there is no way to access the data, even with the assistance of technical support. The database must be discarded and you must create a new database.

Encrypted values can be decrypted with the DECRYPT function. You must use the same key that was specified in the ENCRYPT function. Both of these functions return LONG BINARY values. If you require a different data type, you can use the CAST function to convert the value to the required data type. The example below shows how to use the CAST function to convert a decrypted value to the required data type. See "CAST function [Data type conversion]" [*SQL Anywhere Server - SQL Reference*].

If database users need to access the data in decrypted form, but you do not want them to have access to the encryption key, you can create a view that uses the DECRYPT function. This allows users to access the decrypted data without knowing the encryption key. If you create a view or stored procedure that uses the table, you can use the SET HIDDEN parameter of the ALTER VIEW and ALTER PROCEDURE statements to ensure that users cannot access the encryption key by looking at the view or procedure definition. See "ALTER PROCEDURE statement" [*SQL Anywhere Server - SQL Reference*] and "ALTER VIEW statement" [*SQL Anywhere Server - SQL Reference*].

Column encryption example

The following example uses triggers to encrypt a column that stores passwords in a table called user_info. The user_info table is defined as follows:

```
CREATE TABLE user_info (
   employee_ID INTEGER NOT NULL PRIMARY KEY,
   user_name CHAR(80),
   user_pwd CHAR(80) );
```

Two triggers are added to the database to encrypt the value in the user_pwd column, either when a new user is added or an existing user's password is updated.

- The encrypt_new_user_pwd trigger fires each time a new row is added to the user_info_table:

```
CREATE TRIGGER encrypt_new_user_pwd
BEFORE INSERT
ON user_info
REFERENCING NEW AS new_pwd
FOR EACH ROW
BEGIN
    SET new_pwd.user_pwd=ENCRYPT(new_pwd.user_pwd, '8U3dkA');
END;
```

- The encrypt_updated_pwd trigger fires each time the user_pwd column is updated in the user_info table:

```
CREATE TRIGGER encrypt_updated_pwd
BEFORE UPDATE OF user_pwd
ON user_info
REFERENCING NEW AS new_pwd
FOR EACH ROW
BEGIN
    SET new_pwd.user_pwd=ENCRYPT(new_pwd.user_pwd, '8U3dkA');
END;
```

Add a new user to the database:

```
INSERT INTO user_info
VALUES ( '1', 'd_williamson', 'abc123');
```

If you issue a SELECT statement to view the information in the user_info table, the value in the user_pwd column is binary data (the encrypted form of the password) and not the value abc123 that was specified in the INSERT statement.

If this user's password is changed:

```
UPDATE user_info
SET user_pwd='xyz'
WHERE employee_ID='1';
```

the encrypt_updated_pwd trigger fires and the encrypted form of the new password appears in the user_pwd column.

The original password can be retrieved by issuing the following SQL statement. This statement uses the DECRYPT function and the encryption key to decrypt the data, and the CAST function to convert the value from a LONG BINARY to a CHAR value:

```
SELECT CAST (
   DECRYPT( user_pwd, '8U3dkA' )
   AS CHAR(100))
   FROM user_info
   WHERE employee_ID = '1';
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

See also

- "ENCRYPT function [String]" [*SQL Anywhere Server - SQL Reference*]
- "DECRYPT function [String]" [*SQL Anywhere Server - SQL Reference*]

Table encryption

Table encryption allows you to encrypt tables or materialized views with sensitive data without the performance impact that encrypting the entire database might cause. When table encryption is enabled, table pages for the encrypted table, associated index pages, and temporary file pages are encrypted. The transaction log pages that contain transactions on encrypted tables are also encrypted.

For information about encrypting materialized views, see "Encrypt and decrypt materialized views" [*SQL Anywhere Server - SQL Usage*].

To encrypt tables in your database, you must have table encryption enabled. Enabling table encryption must be done at database initialization. To see whether table encryption is enabled, query the EncryptionScope database property using the DB_PROPERTY function, as follows:

```
SELECT DB_PROPERTY( 'EncryptionScope' );
```

If the return value is TABLE, table encryption is enabled.

To see the encryption algorithm in effect for table encryption, query the Encryption database property using the DB_PROPERTY function, as follows:

```
SELECT DB_PROPERTY( 'Encryption' );
```

For a list of supported encryption algorithms, see "Encrypting and decrypting a database" on page 1130.

Performance impact of table encryption

For encrypted tables, each table page is encrypted when written to the disk, and is decrypted when read in from the disk. This process is invisible to applications. However, there may be a slight negative impact on performance when reading from, or writing to, encrypted tables. Encrypting or decrypting existing tables can take a long time, depending on the size of the table.

Index pages for indexes on columns in an encrypted table are also encrypted, as are transaction log pages containing transactions on the encrypted table, and all pages in the temporary file for the database. All other database and transaction log pages are unencrypted.

Encrypted tables can contain compressed columns. In this case, the data is compressed before it is encrypted.

Encrypting tables does not impact storage requirements.

Starting a database that has table encryption enabled

Starting a database that has table encryption enabled is the same as starting an encrypted database. For example, if the database is started with the -ek option, a key must be specified. If the database is started with the -ep option, you are prompted for the key. See "Initialization utility (dbinit)" on page 799.

Enabling table encryption in the database

Table encryption must be enabled and configured at database creation time. You must re-create the database with table encryption enabled if your database does not have table encryption enabled, or if you have database encryption in effect.

To create a database with table encryption (SQL)

- Create a database with the CREATE DATABASE statement, and specify a key and an encryption algorithm.

  The following command creates the database *new.db* with strong encryption enabled for tables using the key abc, and the AES256_FIPS encryption algorithm:

  ```
  CREATE DATABASE 'new.db'
     ENCRYPTED TABLE
     KEY 'abc'
     ALGORITHM 'AES256_FIPS';
  ```

  Later, when you encrypt a table in this database, the AES256_FIPS algorithm and abc key are used.

To create a database with table encryption (command line)

- Create a database with the dbinit -et and -ek options, and specify a key and an encryption algorithm.

  The following command creates the database *new.db* with strong encryption enabled for tables using the key abc and the AES256_FIPS encryption algorithm:

  ```
  dbinit new.db -et -ek abc -ea AES256_FIPS
  ```

  Later, when you encrypt a table in this database, the AES256_FIPS algorithm and abc key are used.

To create a database with table encryption using an existing database (SQL)

- Create an encrypted copy of the database with the CREATE ENCRYPTED TABLE DATABASE statement, and specify a key.

  The following example creates a database called contacts2 from an existing database called contacts1. The new database supports encrypted tables.

  ```
  CREATE ENCRYPTED TABLE DATABASE 'contacts2.db'
     FROM 'contacts1.db'
     KEY 'Sd8f6654'
     OLD KEY 'Sc8e5543';
  ```

  Later, when you encrypt a table in this database, the AES algorithm and Sd8f6654 key are used.

Encrypting a table

To encrypt tables in your database, table encryption must already be enabled in the database. See "Enabling table encryption in the database" on page 1140.

When you encrypt a table, the encryption algorithm and key that were specified at database creation time are used.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

To encrypt a table at table creation (SQL)

- Create a table using the ENCRYPTED clause of the CREATE TABLE statement.

  The following command creates an encrypted table named MyEmployees:

```
CREATE TABLE MyEmployees (
 MemberID CHAR(40),
 CardNumber INTEGER )
ENCRYPTED;
```

To encrypt a table after it has been created (SQL)

- Encrypt a table with the ENCRYPTED clause of the ALTER TABLE statement.

  The following statements create a table called MyEmployees2 and then encrypt it.

```
CREATE TABLE MyEmployees2 (
   MemberID CHAR(40),
   CardNumber INTEGER );
ALTER TABLE MyEmployees2
   ENCRYPTED;
```

See also

- "Encrypting and decrypting a database" on page 1130
- "Initialization utility (dbinit)" on page 799
- "Creating an encrypted database" on page 1132
- "Create a database (Sybase Central)" on page 9
- "ALTER TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE ENCRYPTED DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE DECRYPTED DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]

Keeping your Windows Mobile database secure

This section describes SQL Anywhere features that help make your Windows Mobile database secure. In particular, this section describes auditing, database encryption, and presents overviews of other security features, providing links to where you can find more information.

Many of the SQL Anywhere security features for Windows desktop platforms are supported on Windows Mobile, such as database file encryption and simple communication encryption, or have modified support, such as the Log Translation utility.

Databases running on Windows Mobile use the same user identification and authorization features as databases running on Windows desktop platforms. These features control who can access the database and what actions those users can perform. See "Controlling database access" on page 1117.

Windows Mobile device security

If you are storing sensitive data on your Windows Mobile device, you may want to use the security features provided for your Windows Mobile device.

For more information about available security features, see the User's Manual provided with your Windows Mobile device.

Database server options

Server options allow you to control who can perform certain operations on the server.

These options are set in the Options field of the **Server Startup Options** window when you start the database on your Windows Mobile device.

For more information, see "Controlling permissions from the command line" on page 48.

For information about setting options on Windows Mobile, see "Specifying server options on Windows Mobile" on page 363.

Auditing

This feature uses the transaction log to maintain a detailed record of actions on the database.

The Log Translation utility (dbtran) is used to translate the information stored in the transaction log, including auditing information. The dbtran utility is not supported on Windows Mobile, so you cannot translate a log stored on a Windows Mobile device. Copy the transaction log file to your PC to use this utility.

For more information, see "Auditing database activity" on page 1123.

Database encryption on Windows Mobile

Database encryption features allow you to choose the level of database encryption. You can choose to secure your database either with simple encryption, or with strong encryption. SQL Anywhere supports both simple and strong encryption on Windows Mobile.

Simple encryption This level of encryption is equivalent to obfuscation and makes it more difficult for someone using a disk utility to look at the file to decipher the data in your database. Simple encryption does not require a key to encrypt the database.

Simple encryption technology is supported in previous versions of SQL Anywhere.

Strong encryption This level of encryption obfuscates the information contained in your database and transaction log files so they cannot be deciphered simply by looking at the files using a disk utility. Strong encryption renders the database completely inaccessible without the key. On Windows Mobile, the AES_FIPS and AES256_FIPS algorithms are only supported with ARM processors.

For more information, see "Encrypting and decrypting a database" on page 1130.

Communication encryption and Windows Mobile

You can encrypt client/server communications for greater security as they pass over the network. SQL Anywhere provides two types of communication encryption: simple and strong.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Simple communication encryption accepts communication packets that are encrypted with simple encryption. This level of communication encryption is supported on all platforms, including Windows Mobile and on previous versions of SQL Anywhere.

Strong communication encryption is not available on Windows Mobile.

For more information about encrypting communications, see "Encryption (ENC) connection parameter" on page 287.

Transport-layer security

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

Transport-layer security, an IETF standard protocol, secures client/server communications using digital certificates and public-key cryptography. Transport-layer security enables encryption, tamper detection, and certificate-based authentication.

You can use transport-layer security to:

- Secure communications between the SQL Anywhere database server and client applications.
- Secure communications between the MobiLink server and MobiLink clients.
- Set up a secure SQL Anywhere web server.

Secure communication begins with an exchange of messages (a handshake) including:

- **Server authentication** Transport-layer security uses server certificates to establish and maintain a secure connection. You create unique certificate files for each server. You can use server authentication for SQL Anywhere client/server communication or for MobiLink synchronization:
  - For SQL Anywhere client/server communication, a database client verifies the identity of a SQL Anywhere database server.
  - For MobiLink synchronization, a MobiLink client (SQL Anywhere or UltraLite) verifies the identity of a MobiLink server.

Efficiency

The transport-layer security protocol uses a combination of public-key and symmetric key encryption. Public-key encryption provides better authentication techniques, but is computationally intensive. Once a secure connection is established, the client and server use a highly efficient symmetric cipher with 128-bit key size for the rest of their communication.

Certificates

SQL Anywhere includes a tool called createcert that allows you to create X.509 certificate files for transport-layer security. However, if you need to verify the existence of third-party certificates, or if you need more secure certificates, you can purchase the certificates from certificate authorities.

Database file encryption

For information about database file encryption, see:

- SQL Anywhere databases: "Encrypting and decrypting a database" on page 1130
- UltraLite databases: "Securing UltraLite databases" [*UltraLite - Database Management and Reference*]

TLS support

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

This topic details the support for RSA, ECC, and FIPS encryption.

RSA encryption

RSA encryption is provided free with SQL Anywhere and can be used for client/server communication, synchronization, and web services. The free version is not FIPS-certified. To implement FIPS-certified RSA encryption, you need a separate license.

For a list of supported platforms for RSA, see http://www.sybase.com/detail?id=1061806.

ECC encryption

To implement ECC encryption, you need a separate license.

For a list of supported platforms for ECC, see http://www.sybase.com/detail?id=1061806.

FIPS-approved encryption

FIPS is only available for RSA encryption. (ECC is not yet covered by the FIPS program.)

FIPS technology requires a separate license. See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

For a list of supported platforms for FIPS, see http://www.sybase.com/detail?id=1061806.

FIPS-approved encryption technology

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

You can use FIPS-certified security algorithms to encrypt your database files, or to encrypt communications for database client/server communication, web services, and MobiLink client/server communication.

Federal Information Processing Standard (FIPS) 140-2 specifies requirements for security algorithms. FIPS 140-2 is granted by the American and Canadian governments through the National Institute of Standards and Testing (NIST) and the Canadian Communications Security Establishment (CSE).

SQL Anywhere uses two FIPS-certified modules for encryption, both from Certicom. On Windows (desktop and Windows Mobile) and Unix platforms, SQL Anywhere uses Certicom Security Builder GSE (FIPS Module v2.0). This is number 542 on the page http://csrc.nist.gov/cryptval/140-1/140val-all.htm.

Enforcing FIPS

Optionally, you can enforce the use of FIPS with a FIPS option. When you set the FIPS option to on, all secure communications must be over FIPS-approved channels. If someone tries to use non-FIPS RSA, it is automatically upgraded to FIPS RSA. If ECC is selected, an error is reported (ECC does not support FIPS). You must set the FIPS option for each computer on which you want FIPS to be enforced. SQL Anywhere and MobiLink servers have a -fips command line option, and clients have a fips option that can be set with the encryption parameter.

For information about encrypting SQL Anywhere database files with FIPS technology, see “Strong encryption” on page 1131.

Setting up transport-layer security

The following steps provide an overview of the tasks required to set up transport-layer security.

1. Obtain digital certificates.

   You need identity files and certificate files. The server identity file contains the server's private key and should be stored securely with the database or MobiLink server. You distribute the server certificate file to your clients.

   You can buy certificates from a certificate authority. SQL Anywhere also provides functionality to create certificates, which is especially useful for development and testing. See “Creating digital certificates” on page 1146.

2. If you are setting up transport-layer security for SQL Anywhere client/server applications:

   - **Start the SQL Anywhere database server with transport-layer security** Use the -ec database server option to specify the type of security, the server identity file name, and the password to protect the server's private key.

     If you also want to allow unencrypted connections over shared memory, specify the -es option.

     See “Starting the database server with transport-layer security” on page 1152.

- **Configure client applications to use transport-layer security** Specify the path and file name of trusted certificates using the Encryption connection parameter [ENC].

  See "Configuring client applications to use transport-layer security" on page 1154.

3. If you are setting up transport-layer security for SQL Anywhere web services:

   - **Start the SQL Anywhere database server with transport-layer security** Use the -xs database server option to specify the type of security, the server identity file name, and the password to protect the server's private key.

   - **Configure browsers or other web clients to trust certificates** See "Encrypting SQL Anywhere web services" on page 1156.

4. If you are setting up transport-layer security for MobiLink synchronization:

   - **Start the MobiLink server with transport-layer security** Use the mlsrv12 -x option to specify the security stream, the server identity file name, and the password to protect the server's private key.

     See "Starting the MobiLink server with transport-layer security" on page 1158.

   - **Configure MobiLink clients to use transport-layer security** Supply the appropriate security or network protocol options with the MobiLink synchronization client utility (dbmlsync) or UltraLite application. Specify the security stream and trusted server certificate file names.

     See "Configuring MobiLink clients to use transport-layer security" on page 1160.

Other resources for getting started

You can post questions on the newsgroups:

- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

Creating digital certificates

You need digital certificates to set up transport-layer security. You can obtain certificates from a certificate authority, or you can create them using SQL Anywhere functionality.

SQL Anywhere Certificate Creation utility

You can use the SQL Anywhere Certificate Creation utility, createcert, to generate X.509 certificate files using RSA or ECC. See "Certificate Creation utility (createcert)" on page 775.

SQL Anywhere Certificate Viewer utility

You can use the SQL Anywhere Certificate Viewer utility, viewcert, to read X.509 certificates using RSA or ECC. See "Certificate Viewer utility (viewcert)" on page 778.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Certificates for server authentication

You can follow the same process to create certificate files for server authentication. In each case, you create an identity file and a certificate file.

For server authentication, you create a server identity file and a certificate file to distribute to clients.

Certificate configurations

The certificate can be self-signed or signed by a commercial or enterprise Certificate Authority.

- **Self-signed certificates** Self-signed server certificates can be used for simple setups. See "Self-signed root certificates" on page 1147.

- **Enterprise root certificates** An enterprise root certificate can be used to sign server certificates to improve data integrity and extensibility for multi-server deployments.

  - You can store the private key used to sign server certificates in a secure central location.
  - For server authentication, you can add MobiLink or database servers without reconfiguring clients.

  See "Certificate chains" on page 1148.

- **Commercial Certificate Authorities** You can use a third-party Certificate Authority instead of an enterprise root certificate. Commercial Certificate Authorities have dedicated facilities to store private keys and create high-quality server certificates.

  See "Certificate chains" on page 1148, and "Globally-signed certificates" on page 1150.

Self-signed root certificates

Self-signed root certificates can be used for simple setups involving a single MobiLink or database server.

Tip
Use enterprise level certificate chains or commercial certificate authorities if you require multiple server identity files. Certificate authorities provide extensibility and a higher level of certificate integrity with dedicated facilities to store root private keys.

For more information about setting up certificate chains, see "Certificate chains" on page 1148.

- **Certificate** For server authentication certificates, the self-signed certificate is distributed to clients. It is an electronic document including identity information, the public key of the server, and a self-signed digital signature.

- **Identity file** For server authentication certificates, the identity file is stored securely with a MobiLink or database server. It is a combination of the self-signed certificate (that is distributed to clients) and the corresponding private key. The private key gives the MobiLink or database server the ability to decrypt messages sent by the client in the initial handshake.

See also

- “Server authentication” on page 1160
- “Starting the database server with transport-layer security” on page 1152
- “Certificate Creation utility (createcert)” on page 775

Certificate chains

If you require multiple identity files, you can improve security and extensibility by using certificate chains instead of self-signed certificates. Certificate chains require a Certificate Authority or an enterprise root certificate to sign identities.

See “Self-signed root certificates” on page 1147.

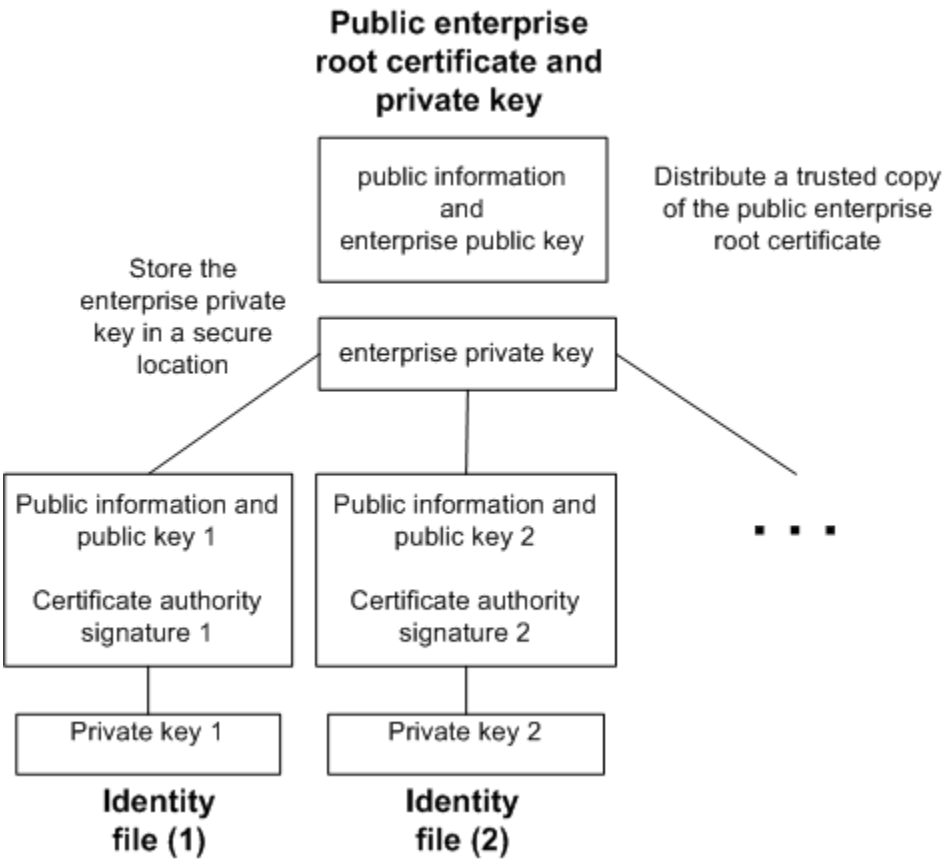
Benefits of using certificate chains

Certificate chains provide the following advantages:

- **Extensibility** For server authentication, you can configure clients to trust any certificate signed by an enterprise root certificate or Certificate Authority. If you add a new MobiLink or database server, clients do not require a copy of the new certificate.
- **Security** The enterprise root certificate's private key is not in the identity file. Storing the root certificate's private key in a high-security location, or using a Certificate Authority with dedicated facilities, protects the integrity of server authentication.

The following diagram provides the basic enterprise root certificate architecture.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0



Using certificates in a multi-server environment

To create certificates used in a multi-server environment:

- Generate a public enterprise root certificate and enterprise private key.

  Store the enterprise private key in a secure location, preferably a dedicated facility.

  For server authentication, you distribute the public enterprise root certificate to clients.

- Use the enterprise root certificate to sign identities.

  Use the public enterprise root certificate and enterprise private key to sign each identity. For server authentication, the identity file is used for the server.

You can also use a third-party Certificate Authority to sign your server certificates. Commercial Certificate Authorities have dedicated facilities to store private keys and create high-quality server certificates.

See also

- "Certificate Creation utility (createcert)" on page 775
- "Globally-signed certificates" on page 1150

Enterprise root certificates

Enterprise root certificates improve data integrity and extensibility for multi-server deployments.

- You can store the private key used to create trusted certificates in a dedicated facility.
- For server authentication, you can add servers without reconfiguring clients.

To set up enterprise root certificates, you create the enterprise root certificate and the enterprise private key that you use to sign identities.

For information about creating server certificates, see "Signed identity files" on page 1150.

For information about generating enterprise root certificates, see "Globally-signed certificates" on page 1150.

Signed identity files

You can use an enterprise root certificate to sign server identity files.

For server authentication, you generate identity files for each server. Since these certificates are signed by an enterprise root certificate, you use the createcert -s option.

For information about generating signed identity files, see "Certificate Creation utility (createcert)" on page 775.

Globally-signed certificates

A commercial Certificate Authority is an organization that is in the business of creating high-quality certificates and using these certificates to sign your certificate requests.

Globally-signed certificates have the following advantages:

- For inter-company communication, common trust in an outside, recognized authority may increase confidence in the security of the system. A Certificate Authority must guarantee the accuracy of the identification information in any certificate that it signs.
- Certificate Authorities provide controlled environments and advanced methods to generate certificates.
- The private key for the root certificate must remain private. Your organization may not have a suitable place to store this crucial information, whereas a Certificate Authority can afford to design and maintain dedicated facilities.

Setting up globally-signed certificates

To set up globally signed identity files, you:

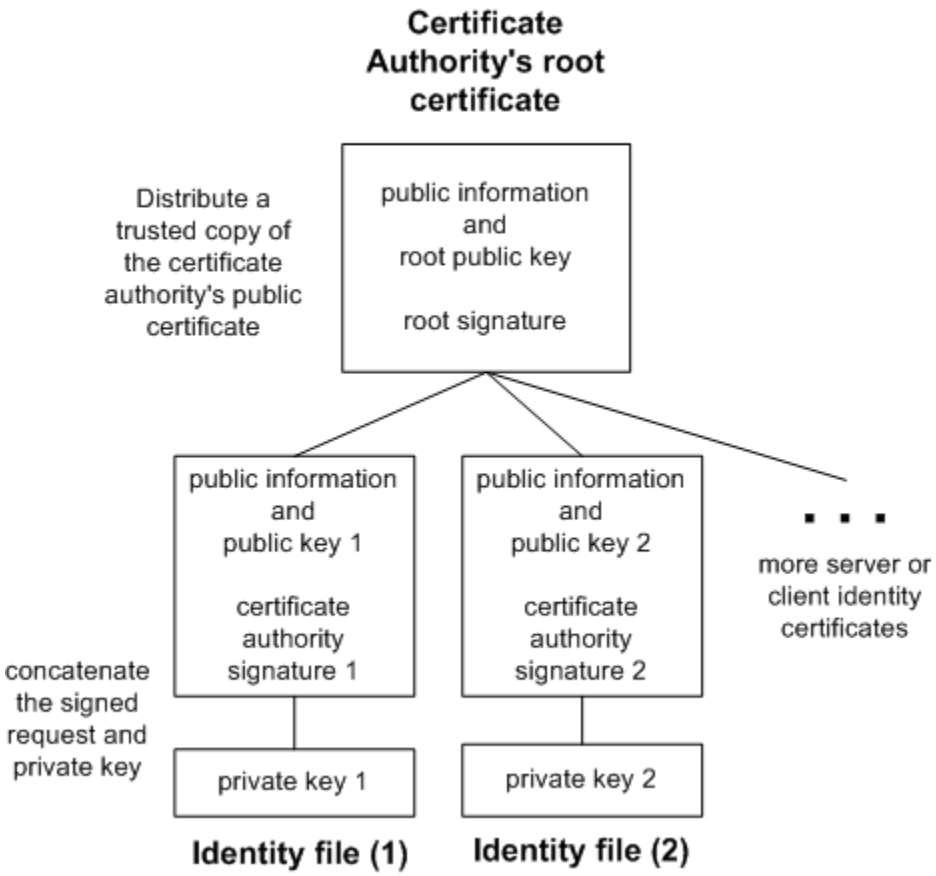
 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- Create a certificate request using the createcert utility with the -r option. See "Certificate Creation utility (createcert)" on page 775.
- Use a Certificate Authority to sign each request. You can combine the signed request with the corresponding private key to create the server identity file.

Globally-signing enterprise root certificates
You might be able to globally-sign an enterprise root certificate. This is only applicable if your Certificate Authority generates certificates that can be used to sign other certificates.

Using globally signed identity files

You can use globally-signed certificates directly as server identity files. The following diagram shows the configuration for multiple identity files:



You reference the server identity file and the password for the private key on the dbsrv12 or mlsrv12 command line.

See also

- SQL Anywhere: "Starting the database server with transport-layer security" on page 1152
- MobiLink: "Starting the MobiLink server with transport-layer security" on page 1158

Setting up clients to trust the certificate authority's certificate

For server authentication, you must ensure that clients contacting your server trust the root certificate in the chain. For globally-signed certificates, the root certificate is the Certificate Authority's certificate.

Certificate field verification
When using a globally-signed certificate, each client must verify field values to avoid trusting certificates that the same Certificate Authority has signed for other clients.

See "Verifying certificate fields" on page 1160.

For more information about configuring MobiLink clients to trust server certificates, see "Configuring MobiLink clients to use transport-layer security" on page 1160.

For more information about configuring the database server to use transport-layer security, see "Starting the database server with transport-layer security" on page 1152.

For more information about using globally-signed certificates to establish trust, see "Globally-signed certificates" on page 1150.

Encrypting SQL Anywhere client/server communications

You can encrypt SQL Anywhere client/server communication using transport-layer security.

See also

- "Encrypting SQL Anywhere web services" on page 1156

Starting the database server with transport-layer security

To start the database server with transport-layer security, supply the server identity file name and the password protecting the server's private key.

For an overview of the steps required to set up transport-layer security, see "Setting up transport-layer security" on page 1145.

Use the -ec database server option to specify the identity and identity_password parameters. If you want to allow unencrypted connections over shared memory, you must also specify the -es option.

Following is the syntax of a partial dbsrv12 command line:

-ec tls(
  tls_type=*cipher***;**

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

identity=*server-identity-filename***;**
identity_password=*password* **)**
-x tcpip

- ***cipher*** The cipher to use. The cipher can be **rsa** or **ecc** for RSA and ECC encryption, respectively. For FIPS-approved RSA encryption, specify **tls_type=rsa;fips=y**. RSA FIPS uses a separate approved library, but is compatible with SQL Anywhere 9.0.2 or later clients using RSA.

  For a list of supported platforms for FIPS, see http://www.sybase.com/detail?id=1061806.

  The cipher must match the encryption (ECC or RSA) used to create your certificates.

  For information about enforcing the FIPS-approved algorithm, see "-fips dbeng12/dbsrv12 server option" on page 182.

- ***server-identity-filename*** The path and file name of the server identity file. If you are using FIPS-approved RSA encryption, you must generate your certificates using the RSA cipher.

  An identity file contains the public certificate and its private key. For certificates that are not self signed, the identity file also contains all the signing certificates.

  For more information about creating the server certificate, which can be self-signed, or signed by a Certificate Authority or enterprise root certificate, see "Creating digital certificates" on page 1146.

- ***password*** The password for the server private key. You specify this password when you create the server certificate.

You can also start the database server with simple encryption. Simple encryption makes it more difficult for someone using a packet sniffer to read the network packets sent between the client and the server, but does not assure data integrity or provide server authentication.

See "-ec dbeng12/dbsrv12 server option" on page 176, and "-es dbeng12/dbsrv12 server option" on page 180.

You specify the TCP/IP protocol using the -x database server option. See "-x dbeng12/dbsrv12 server option" on page 236.

Example

The following example (entered all on one line) uses the -ec database server option to specify ECC security, the server identity file, and the password protecting the server's private key:

```
dbsrv12 -ec tls( tls_type=ecc;identity=c:\test
\serv1_ecc.id;identity_password=mypwd )
 -x tcpip c:\test\secure.db
```

You can hide the command line options, including passwords, using a configuration file and the File Hiding utility (dbfhide). See "File Hiding utility (dbfhide)" on page 794, and "@data dbeng12/dbsrv12 server option" on page 157.

Configuring client applications to use transport-layer security

You can configure SQL Anywhere client applications to use transport-layer security. Using a set of encryption connection parameters, you specify trusted certificates, the type of encryption, and the network protocol.

For an overview of the steps required to set up transport-layer security, see "Setting up transport-layer security" on page 1145.

Server authentication

Server authentication allows a remote client to verify the identity of a database server. Digital signatures and certificate field verification work together to achieve server authentication.

Digital signatures

A database server certificate contains one or more digital signatures used to maintain data integrity and protect against tampering. Following are the steps used to create a digital signature:

- An algorithm performed on a certificate generates a unique value or hash.
- The hash is encrypted using a signing certificate's or Certificate Authority's private key.
- The encrypted hash, called a digital signature, is embedded in the certificate.

A digital signature can be self-signed or signed by an enterprise root certificate or Certificate Authority.

When a client application contacts a database server, and each is configured to use transport-layer security, the server sends the client a copy of its certificate. The client decrypts the certificate's digital signature using the server's public key included in the certificate, calculates a new hash of the certificate, and compares the two values. If the values match, this confirms the integrity of the server's certificate.

If you are using FIPS-approved RSA encryption, you must generate your certificates using RSA.

For more information about self-signed certificates, see "Self-signed root certificates" on page 1147.

For more information about enterprise root certificates and Certificate Authorities, see "Certificate chains" on page 1148.

Verifying certificate fields

When using a globally signed certificate, each client must verify certificate field values to avoid trusting certificates that the same Certificate Authority has signed for other clients. This is resolved by requiring your clients to test the value of fields in the identity portion of the certificate. A Certificate Authority must guarantee the accuracy of the identification information in any certificate that it signs.

For more information about globally signed certificates, see "Globally-signed certificates" on page 1150.

When creating a certificate using the createcert utility, you enter values for the organization, organizational unit, and common name fields. You verify these fields using corresponding client

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

connection parameters. It is strongly recommended that you verify certificate fields if you are using a third-party Certificate Authority to globally sign certificates.

- **Organization** The organization field corresponds to the certificate_company encryption protocol option. See "certificate_company protocol option" on page 315.
- **Organizational unit** The organizational unit field corresponds to the certificate_unit encryption protocol option. See "certificate_unit protocol option" on page 317.
- **Common name** The common name field corresponds to the certificate_name encryption protocol option. See "certificate_name protocol option" on page 316.

For more information about client-side encryption connection parameters, see "Encryption (ENC) connection parameter" on page 287.

Using the trusted_certificates protocol option

This is the only required protocol option if TLS is specified in the Encryption connection parameter. Clients use the trusted_certificates encryption protocol option to specify trusted database server certificates. The trusted certificate can be a server's self-signed certificate, a public enterprise root certificate, or a certificate belonging to a commercial Certificate Authority.

See also

- "trusted_certificates protocol option" on page 339
- "Creating digital certificates" on page 1146

Establishing a client connection using transport-layer security

To set up client applications to use transport-layer security, use the Encryption [ENC] connection parameter in your connection string. The connection string takes the following form (which must be written all on one line):

Encryption=tls(
tls_type=*cipher***;**
[fips={ y | n };]
trusted_certificates=*public-certificate*
[certificate_company=*organization***;]**
[certificate_name=*common-name***;]**
[certificate_unit=*organization-unit* **])**

- ***cipher*** can be **rsa** or **ecc** for RSA and ECC encryption, respectively. The default is **rsa**. For FIPS-approved RSA encryption, specify **tls_type=rsa;fips=y**. RSA FIPS uses a separate approved library, but is compatible with SQL Anywhere 9.0.2 or later database servers using RSA. You cannot specify **fips=y** with **tls_type=ecc**.

  The connection fails if the cipher does not match the encryption (RSA or ECC) used to create your certificates.

- ***public-certificate*** is the path and file name of a file that contains one or more trusted certificates. If you are using FIPS-approved RSA encryption, you must generate your certificates using RSA. See "trusted_certificates protocol option" on page 339.

- ***organization*** forces the client to accept server certificates only when the Organization field on the certificate matches this value. See "certificate_company protocol option" on page 315.

- ***common-name*** forces the client to accept server certificates only when the Common Name field on the certificate matches this value. See "certificate_name protocol option" on page 316.

- ***organization-unit*** forces the client to accept server certificates only when the Organization Unit field on the certificate matches this value. See "certificate_unit protocol option" on page 317.

For more information about trusted_certificates and other client security parameters, see "Verifying certificate fields" on page 1154 and "Using the trusted_certificates protocol option" on page 1155.

For more information about creating or obtaining the certificate, see "Creating digital certificates" on page 1146.

For more information about the encryption connection parameter, see "Encryption (ENC) connection parameter" on page 287.

Example

The following example uses the trusted_certificates encryption connection parameter to specify the certificate, *public_cert.crt*.

```
"UID=DBA;PWD=sql;Host=myhost;Server=myserver;
ENC=tls(tls_type=ecc;trusted_certificates=public_cert.crt)"
```

The following example uses the trusted_certificates encryption connection parameter to specify the certificate, *public_cert.crt*, and verifies certificate fields using the certificate_unit and certificate_name encryption connection parameters.

```
"UID=DBA;PWD=sql;Host=myhost;Server=myserver;
ENC=tls(tls_type=ecc;trusted_certificates=public_cert.crt;
certificate_unit=test_unit;certificate_name=my_certificate)"
```

Encrypting SQL Anywhere web services

The SQL Anywhere web server supports HTTPS connections using SSL version 3.0 and TLS version 1.0.

To set up transport-layer security for SQL Anywhere web services, perform the following steps:

- **Obtain digital certificates** You need database server certificate files and identity files. Certificates (which can be Certificate Authority certificates) are distributed to browsers or web clients. server identity files are stored securely with your SQL Anywhere web server.

  For general information about creating digital certificates, including information about using Certificate Authorities, see "Creating digital certificates" on page 1146.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Start the web server with transport-layer security** Use the -xs database server option to specify HTTPS, the server identity file, and the password to protect the private key.

  Following is the syntax of a partial dbsrv12 command line.

  -xs *protocol***(**
  [fips={ y | n };]
  identity=*server-identity-filename***;**
  identity_password=*password***;...)** ...

  - ***protocol*** can be **https**, or **https** with **fips=y** for FIPS-approved RSA encryption. FIPS-approved HTTPS uses a separate approved library, but is compatible with HTTPS.

    Note
    The Mozilla Firefox browser can connect when FIPS-approved HTTPS is used. However, the cipher suite used by FIPS-approved HTTPS is not supported by most versions of the Internet Explorer, Opera, or Safari browsers—if you are using FIPS-approved HTTPS, these browsers may not be able to connect.

    For information about enforcing the FIPS-approved algorithm, see "-fips dbeng12/dbsrv12 server option" on page 182.

  - ***server-identity-filename*** The path and file name of the server identity. For HTTPS, you must use an RSA certificate.

  - ***password*** The password for the server private key. You specify this password when you create the server certificate.

  For more information about the -xs server option, see "-xs dbeng12/dbsrv12 server option" on page 241.

  For more information about the identity and identity_password parameters, see:

  - "Identity protocol option" on page 323
  - "Identity_Password protocol option" on page 324

- **Configure web clients** Configure browsers or other web clients to trust certificates. The trusted certificate can be self-signed, an enterprise root, or a Certificate Authority certificate.

  For general information about creating digital certificates, including information about using Certificate Authorities, see "Creating digital certificates" on page 1146.

Encrypting MobiLink client/server communications

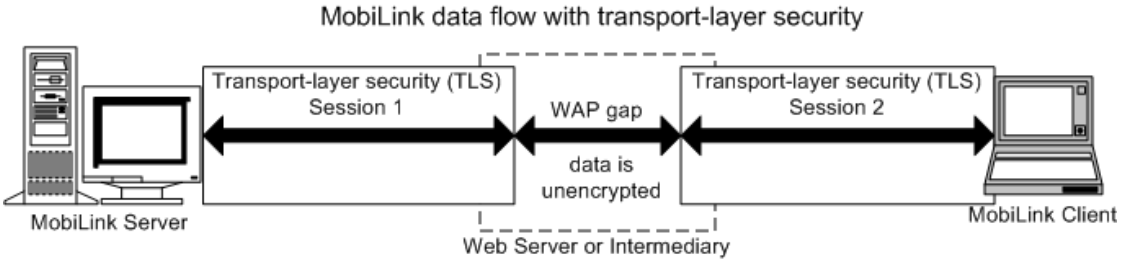
You can encrypt MobiLink client/server communication using transport-layer security.

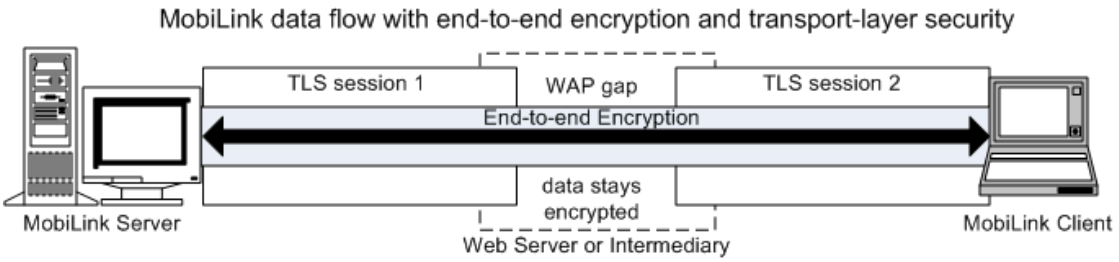
End-to-end encryption

End-to-end encryption occurs when data is encrypted at the point of origin and decrypted at the final destination. There is no point during transmission that the data is unencrypted.

MobiLink TLS is sometimes only used to encrypt data up to an intermediary (for example, encryption/decryption hardware) between the client and server. At the intermediary, the data would be decrypted and then encrypted again by the intermediary for the rest of the journey. Notably, this happens when synchronizing via HTTPS through a Web server. The brief interval when the data is unencrypted in the intermediary is sometimes called the Wireless Application Protocol gap or WAP Gap.

Within a corporation, a WAP gap is often acceptable when the intermediary is within corporate control. However, in a third-party hosted environment where data from different corporations is going through the same WAP gap, sensitive data may be exposed. End-to-end encryption prevents any intermediary from accessing the data because the synchronization stream is encrypted from start to finish, and may optionally be encrypted once more with TLS.





Starting the MobiLink server with transport-layer security

To start the MobiLink server with transport-layer security, supply the identity file and the identity password protecting the server's private key.

For an overview of the steps required to set up transport-layer security, see “Setting up transport-layer security” on page 1145.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Securing the MobiLink server over TCP/IP and HTTPS

Use the mlsrv12 -x server option to specify an identity and an identity password. Following is a partial mlsrv12 command line (which must be written on one line):

-x *protocol***(**
tls_type=*cipher***;**
fips={ y | n };
identity=*identity-file***;**
identity_password=*password***;...** **)**

- ***protocol*** The protocol to use. It can be **https** or **tls**. The **tls** protocol is TCP/IP with TLS.
- ***cipher*** The cipher to use. It can be **rsa** or **ecc** for RSA and ECC encryption, respectively. The cipher must match the encryption used to create your identity.
- **fips** Indicates whether to use FIPS. FIPS can only be used with RSA encryption. RSA FIPS uses separate FIPS 140-2 certified software from Certicom. Servers using FIPS are compatible with clients not using FIPS and vice versa. RSA FIPS can be used for SQL Anywhere clients on any supported 32-bit Windows platform or Solaris, or for UltraLite clients on Unix or any supported 32-bit Windows platform including Windows Mobile.
- ***identity-file*** The path and file name of the identity file, which contains the server's private key, the server's certificate, and, optionally, the certificates signed by the Certificate Authority.

  For information about creating the server certificate, which can be self-signed, or signed by a Certificate Authority or enterprise root certificate, see "Creating digital certificates" on page 1146.
- ***password*** The password for the server private key. You specify this password when you create the server identity.

See "-x mlsrv12 option" [*MobiLink - Server Administration*].

Examples

The following example specifies the type of security (RSA), the server identity file, and the identity password protecting the server's private key on the mlsrv12 command line:

```
mlsrv12 -c "dsn=my_cons"
 -x tls(tls_type=rsa;identity=c:\test\serv_rsa1.crt;identity_password=pwd)
```

The following example specifies an ECC identity on the mlsrv12 command line:

```
mlsrv12 -c "dsn=my_cons"
 -x tls(tls_type=ecc;identity=c:\test\serv_ecc1.crt;identity_password=pwd)
```

The following example is similar to the previous, except that there is a space in the identity file name:

```
mlsrv12 -c "dsn=my_cons"
 -x "tls(tls_type=rsa;identity=c:\Program Files\test
\serv_rsa1.crt;identity_password=pwd)"
```

For more information about the mlsrv12 -x option, see "-x mlsrv12 option" [*MobiLink - Server Administration*].

For more information about creating the server identity file, in this case *serv_ecc1.crt*, see "Creating digital certificates" on page 1146.

You can hide the command line options using a configuration file and the File Hiding utility (dbfhide). See "@data mlsrv12 option" [*MobiLink - Server Administration*].

Configuring MobiLink clients to use transport-layer security

You can configure SQL Anywhere or UltraLite clients to use MobiLink transport-layer security. For each client, you specify trusted certificates, the type of encryption, and the network protocol.

For an overview of the steps required to set up transport-layer security, see "Setting up transport-layer security" on page 1145.

Server authentication

Server authentication allows a remote client to verify the identity of a server. Digital signatures and certificate field verification work together to achieve server authentication.

Digital signatures

A server certificate contains one or more digital signatures used to maintain data integrity and protect against tampering. Following are the steps used to create a digital signature:

- An algorithm performed on a certificate generates a unique value or hash.
- The hash is encrypted using a signing certificate's or Certificate Authority's private key.
- The encrypted hash, called a digital signature, is embedded in the certificate.

A digital signature can be self-signed or signed by an enterprise root certificate or Certificate Authority.

When a MobiLink client contacts a MobiLink server, and each is configured to use transport-layer security, the server sends the client a copy of its certificate. The client decrypts the certificate's digital signature using the server's public key included in the certificate, calculates a new hash of the certificate, and compares the two values. If the values match, this confirms the integrity of the server's certificate.

For more information about self-signed certificates, see "Self-signed root certificates" on page 1147.

For more information about enterprise root certificates and Certificate Authorities, see "Certificate chains" on page 1148.

Verifying certificate fields

When using a globally signed certificate, each client must verify certificate field values to avoid trusting certificates that the same Certificate Authority has signed for other clients. This is resolved by requiring your clients to test the value of fields in the identity portion of the certificate. A Certificate Authority must guarantee the accuracy of the identification information in any certificate that it signs.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

For more information about globally signed certificates, see "Globally-signed certificates" on page 1150.

When creating a certificate using the createcert utility, you enter values for the organization, organizational unit, and common name fields. You verify these fields using corresponding MobiLink client connection parameters.

- **Organization** The organization field corresponds to the certificate_company MobiLink client connection parameter. See "certificate_company" [*MobiLink - Client Administration*].
- **Organizational unit** The organizational unit field corresponds to the certificate_unit MobiLink client connection parameter. See "certificate_unit" [*MobiLink - Client Administration*].
- **Common name** The common name field corresponds to the certificate_name MobiLink client connection parameter. See "certificate_name" [*MobiLink - Client Administration*].

For more information about setting up MobiLink clients, see:

- "Configuring UltraLite clients to use transport-layer security" on page 1163
- "Client security options" on page 1161

For more information about creating digital certificates, see "Creating digital certificates" on page 1146.

Client security options

MobiLink clients (SQL Anywhere and UltraLite) use a common set of connection parameters to configure transport-layer security.

trusted_certificates protocol option

MobiLink clients use the trusted_certificates protocol option to specify trusted MobiLink server certificates. The trusted certificate can be a server's self-signed certificate, a public enterprise root certificate, or the certificate belonging to a commercial Certificate Authority.

See:

- "trusted_certificates" [*MobiLink - Client Administration*]
- "Creating digital certificates" on page 1146

Verifying certificate fields

The certificate_company, certificate_unit, and certificate_name protocol options are used to verify certificate fields, an important step for server authentication. It is strongly recommended that you verify certificate fields if you are using a third-party Certificate Authority to globally-sign certificates.

See:

- "Verifying certificate fields" on page 1160
- "Globally-signed certificates" on page 1150
- "Server authentication" on page 1160

Configuring SQL Anywhere clients to use transport-layer security

This section shows you how to configure SQL Anywhere clients to use transport-layer security over HTTPS or TCP/IP.

Using transport-layer security over TCP/IP and HTTPS

MobiLink transport-layer security is an inherent feature of the MobiLink HTTPS and TCP/IP protocols. To use transport-layer security over HTTPS, specify the trusted_certificates connection parameter using the ADR extended option. Following is the syntax for a partial dbmlsync command line.

-e "ctp=*protocol***;**
adr=[fips={ y | n };]
trusted_certificates=*public-certificate*;
…**"**

- ***protocol*** The protocol to use. It can be **https** or **tls**. The **tls** protocol is TCP/IP using transport-layer security.
- **fips** Indicates whether to use FIPS. FIPS can only be used with RSA encryption. FIPS-approved HTTPS uses separate FIPS 140-2 certified software from Certicom, but is compatible with version 9.0.2 or later MobiLink servers using HTTPS.
- ***public-certificate*** The path and file name of a trusted certificate.

  For HTTPS or FIPS-approved HTTPS, you must use certificates created using RSA encryption.

See also

- "Client security options" on page 1161
- "Creating digital certificates" on page 1146
- "CommunicationAddress (adr) extended option" [*MobiLink - Client Administration*]
- "MobiLink client network protocol option summary" [*MobiLink - Client Administration*]
- "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]

Examples

The following example specifies RSA security over HTTPS. It must all be written on one line:

```
dbmlsync -c "eng=rem1;uid=dba;pwd=mypwd"
  -e "ctp=https;
      adr='trusted_certificates=c:\temp\public_cert.crt;
      certificate_company=Sybase, Inc.;
      certificate_unit=IAS;
      certificate_name=MobiLink'"
```

Alternatively, you can specify the CommunicationAddress extended option using the CREATE SYNCHRONIZATION SUBSCRIPTION or ALTER SYNCHRONIZATION SUBSCRIPTION statement. This method provides the same information, but stores it in the database.

```
CREATE SYNCHRONIZATION SUBSCRIPTION
 TO pub1
```

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

```
FOR user1
ADDRESS 'trusted_certificates=c:\temp\public_cert.crt;
   certificate_company=Sybase, Inc.;
   certificate_unit=IAS;
 certificate_name=MobiLink';
```

The following example specifies RSA security and TCP/IP. It must all be written on one line:

```
dbmlsync -c "eng=rem1;uid=myuid;pwd=mypwd"
   -e "ctp=tls;
       adr='port=3333;
           tls_type=rsa;
           trusted_certificates=c:\test\public_cert.crt;
           certificate_company=Sybase, Inc.;
           certificate_unit=IAS;
           certificate_name=MobiLink'"
```

Alternatively, you can specify the CommunicationAddress extended option using the CREATE SYNCHRONIZATION SUBSCRIPTION or ALTER SYNCHRONIZATION SUBSCRIPTION statement:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
 TO pub1
 FOR user1
 ADDRESS 'port=3333;
    tls_type=rsa;trusted_certificates=public_cert.crt;
       certificate_company=Sybase, Inc.;
       certificate_unit=IAS;
       certificate_name=MobiLink';
```

Configuring UltraLite clients to use transport-layer security

MobiLink transport-layer security is an inherent feature of the MobiLink HTTPS protocol. If you use HTTPS and UltraLite clients, you can specify trusted certificates and certificate fields directly as network protocol options.

For more information about specifying the HTTPS protocol for your UltraLite interface, see "Network protocol options for UltraLite synchronization streams" [*UltraLite - Database Management and Reference*].

For more information about the tls_type synchronization parameter, see "tls_type" [*MobiLink - Client Administration*].

To configure your UltraLite client to use transport-layer security over TCP/IP or HTTPS

1. There are two ways to specify trusted root certificates:
   - **When creating the UltraLite database** See "UltraLite Initialize Database utility (ulinit)" [*UltraLite - Database Management and Reference*].
   - **Using the trusted_certificates protocol option** For details, see Step 3 of this procedure.
2. Specify the TCP/IP or HTTPS protocol for synchronization. The keyword for secure TCP/IP is tls.

   The following example is in C/C++ UltraLite. To specify tls, change https to tls.

```
auto ul_sync_info synch_info;
conn.InitSynchInfo( &synch_info );
synch_info.user_name = UL_TEXT( "50" );
synch_info.version = UL_TEXT( "ul_default" );
...
synch_info.stream = "https";
...
```

3. Specify TCP/IP or HTTPS protocol options.

   The following example is in C/C++ UltraLite. To specify tls, change https to tls.

   ```
   auto ul_sync_info synch_info;
   ...
   synch_info.stream = "https";
   synch_info.stream_parms = TEXT(
         "port=9999;
          certificate_company=Sybase, Inc.;
          certificate_unit=IAS;
          certificate_name=MobiLink");
   ```

   The certificate_company, certificate_unit, and certificate_name protocol options are used to verify certificate fields.

   See "Verifying certificate fields" on page 1160.

   You can also specify the trusted_certificates HTTPS protocol option, which overrides any trusted certificate information embedded in the UltraLite database (Step 1 of this procedure).

   ```
   auto ul_sync_info synch_info;
   ...
   synch_info.stream = "https";
   synch_info.stream_parms = TEXT(
         "port=9999;
          trusted_certificates=\rsaroot.crt;
          certificate_company=Sybase, Inc.;
          certificate_unit=IAS;
          certificate_name=MobiLink");
   ```

   For more information about HTTPS options, see "Network protocol options for UltraLite synchronization streams" [*UltraLite - Database Management and Reference*].

Certificate utilities

Users may typically go to a third party to purchase certificates. These certificate authorities provide their own tools for creating certificates. The following tools may be especially useful to create certificates for development and testing purposes, and can also be used for production certificates.

See:

- "Certificate Creation utility (createcert)" on page 775
- "Certificate Viewer utility (viewcert)" on page 778

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Replication

This section describes how to use SQL Anywhere as an Open Server.

For information about the replication and synchronization technologies supported by SQL Anywhere, see "Comparing synchronization technologies" [*SQL Anywhere 12 - Introduction*].

Using SQL Anywhere as an Open Server

SQL Anywhere can appear as an Open Server to client applications. This feature enables Sybase Open Client applications to connect natively to SQL Anywhere databases.

If you simply want to use a Sybase application with SQL Anywhere, you do not need to know any details of Sybase Open Client, Sybase Open Server, or TDS. However, an understanding of how these components fit together may be helpful for configuring your database and setting up applications. This section explains how the components fit together, but avoids any discussion of the internal features of the components.

Sybase Open Client and Sybase Open Server

SQL Anywhere and other members of the Adaptive Server family act as **Open Servers**. This means you can develop client applications using the **Sybase Open Client** libraries available from Sybase. Sybase Open Client includes both the Client Library (CT-Library) and the older DB-Library interfaces.

For information about developing Sybase Open Client applications for use with SQL Anywhere, see "Sybase Open Client support" [*SQL Anywhere Server - Programming*].

Tabular data stream

Sybase Open Client and Sybase Open Server exchange information using an application protocol called the **tabular data stream** (TDS). All applications built using the Sybase Open Client libraries are also TDS applications because the Sybase Open Client libraries handle the TDS interface. However, some applications (such as jConnect) are TDS applications even though they do not use the Sybase Open Client libraries—they communicate directly using the TDS protocol.

While many Open Servers use the Sybase Open Server libraries to handle the interface to TDS, some applications have a direct interface to TDS of their own. Sybase Adaptive Server Enterprise and SQL Anywhere both have internal TDS interfaces. They appear to client applications as an Open Server, but do not use the Sybase Open Server libraries.

Programming interfaces and application protocols

SQL Anywhere supports two application protocols. Sybase Open Client applications and other Sybase applications such as OmniConnect use TDS. ODBC and embedded SQL applications use a separate application protocol specific to SQL Anywhere.

TDS uses TCP/IP

Application protocols such as TDS sit on top of lower-level communications protocols that handle network traffic. SQL Anywhere supports TDS only over the TCP/IP network protocol. In contrast, the SQL Anywhere-specific application protocol supports several network protocols, and a shared memory protocol designed for same-computer communication.

Sybase applications and SQL Anywhere

The ability of SQL Anywhere to act as an Open Server enables Sybase applications such as OmniConnect to work with SQL Anywhere.

OmniConnect support

Sybase OmniConnect provides a unified view of disparate data within an organization, allowing users to access multiple data sources without having to know what the data looks like or where to find it. In addition, OmniConnect performs heterogeneous joins of data across the enterprise, enabling cross-platform table joins of targets such as IBM DB2, Sybase Adaptive Server Enterprise, Oracle, and VSAM.

Using the Open Server interface, SQL Anywhere can act as a data source for OmniConnect.

Setting up SQL Anywhere as an Open Server

This section describes how to set up a SQL Anywhere server to receive connections from Sybase Open Client applications.

System requirements

There are separate requirements at the client and server for using SQL Anywhere as an Open Server.

Server-side requirements

You must have the following elements at the server side to use SQL Anywhere as an Open Server:

- **SQL Anywhere server components** You must use the network server (*dbsrv12.exe*) if you want to access an Open Server over a network. You can use the personal server (*dbeng12.exe*) as an Open Server only for connections from the same computer.

- **TCP/IP** You must have a TCP/IP protocol stack to use SQL Anywhere as an Open Server, even if you are not connecting over a network.

Client-side requirements

You need the following elements to use Sybase client applications to connect to an Open Server (including SQL Anywhere):

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

- **Sybase Open Client components** The Sybase Open Client libraries provide the network libraries your application needs to communicate via TDS if your application uses Sybase Open Client.

- **jConnect** If your application uses JDBC, you need jConnect and a Java Runtime Environment. SQL Anywhere supports jConnect 5.5 and 6.0.5. To download either version of jConnect, go to http://www.sybase.com/products/informationmanagement/softwaredeveloperkit/jconnect.

- **DSEdit** You need DSEdit, the directory services editor, to make server names available to your Sybase Open Client application. On Unix platforms, this utility is called sybinit.

  DSEdit is not included with SQL Anywhere, but is included with Sybase Open Server software.

Starting the database server as an Open Server

If you want to use SQL Anywhere as an Open Server, you must ensure that you start it using the TCP/IP protocol. By default, the server starts all available communications protocols, but you can limit the protocols started by listing them explicitly in the command. For example, the following commands are both valid:

```
dbsrv12 -x tcpip -n myserver c:\mydata.db
```

You can use the personal database server as an Open Server for communications on the same computer because it supports the TCP/IP protocol.

The server can serve other applications through the TCP/IP protocol or other protocols using the SQL Anywhere-specific application protocol at the same time as serving Sybase Open Client applications over TDS.

Port numbers

Every application using TCP/IP on a computer uses a distinct TCP/IP **port** so that network packets end up at the right application. The default port for SQL Anywhere is port 2638. It is recommended that you use the default port number as SQL Anywhere has been granted that port number by the Internet Assigned Numbers Authority (IANA). If you want to use a different port number, you can specify which one using the ServerPort (PORT) protocol option:

```
dbsrv12 -x tcpip(ServerPort=2629) -n myserver c:\mydata.db
```

You may also need to supply a ServerName if more than one local database server is running, or if you want to connect to a network server.

Sybase Open Client settings

To connect to this server, the interfaces file at the client computer must contain an entry specifying the computer name on which the database server is running, and the TCP/IP port it uses.

For information about setting up the client computer, see “Configuring Sybase Open Server” on page 1168.

Configuring Sybase Open Server

SQL Anywhere can communicate with other Adaptive Servers, Open Server applications, and client software on the network. Clients can talk to one or more servers, and servers can communicate with other servers via remote procedure calls. For products to interact with one another, each needs to know where the others reside on the network. This network service information is stored in the interfaces file.

The interfaces file

The **interfaces file** is usually named *SQL.ini* on Windows operating systems and *interfaces*, or *interfac* on Unix operating systems.

Like an address book, the interfaces file lists the name and address of every database server known to Sybase Open Client applications on your computer. When you use a Sybase Open Client program to connect to a database server, the program looks up the server name in the interfaces file and then connects to the server using the address.

The name, location, and contents of the interfaces file differ between operating systems. Also, the format of the addresses in the interfaces file differs between network protocols.

When you install SQL Anywhere, the installer creates a simple interfaces file that you can use for local connections to SQL Anywhere over TCP/IP. It is the System Administrator's responsibility to modify the interfaces file and distribute it to users so that they can connect to SQL Anywhere over the network.

Using the DSEdit utility

The DSEdit utility is a Windows utility that allows you to configure the interfaces file (*SQL.ini*). The following sections explain how to use the DSEdit utility to configure the interfaces file.

These sections describe how to use DSEdit for those tasks required for SQL Anywhere. It is not complete documentation for the DSEdit utility.

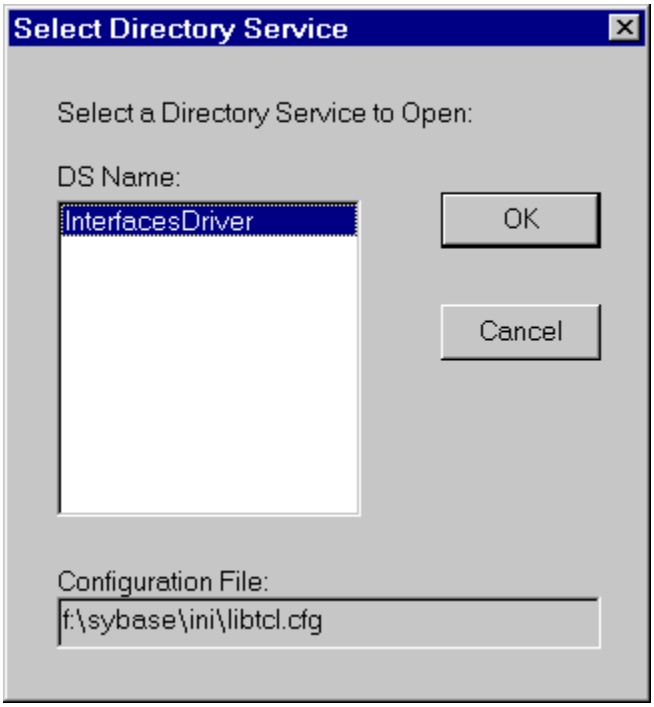
For more information about DSEdit, see the *Configuration Guide* for your platform, included with other Sybase products.

Starting DSEdit

The DSEdit executable is located in the *SYBASE\bin* directory, which is added to your path on installation.

When you start DSEdit, the **Select Directory Service** window appears.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0



Opening a directory services session

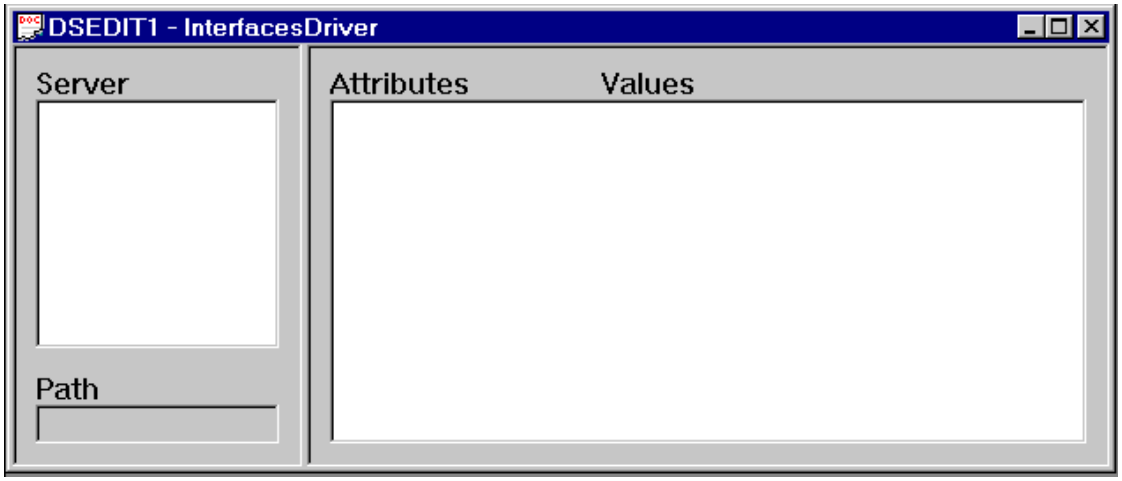
The **Select Directory Service** window allows you to open a session with a directory service. You can open a session to edit the interfaces file (*SQL.ini*), or any directory service that has a driver listed in the *libtcl.cfg* file.

To open a session

- In the **DS Name** list, click the local name of the directory service you want to connect to and click **OK**.

  For SQL Anywhere, select **InterfacesDriver**.

SYBASE environment variable must be set
The DSEdit utility uses the SYBASE environment variable to locate the *libtcl.cfg* file. If the SYBASE environment variable is incorrect, DSEdit cannot locate the *libtcl.cfg* file.



You can add, modify, or delete entries for servers, including SQL Anywhere servers, in the **InterfacesDriver** window.

Adding a server entry

To add a server entry

1. From the **Server Object** menu, choose **Add**.
2. In the **Server Name** box, type a server name and click **OK**.

   The server name entry must match the database name you plan to connect to. The server address is used to identify and locate the server. The **Server Name** field is an identifier for Sybase Open Client. For SQL Anywhere, if the database server has more than one database loaded, the DSEdit server name entry identifies which database to use.

   The server entry appears in the **Server** box. To specify the attributes of the server, you must modify the entry.

Adding or changing the server address

Once you have entered a **Server Name**, you need to modify the **Server Address** to complete the interfaces file entry.

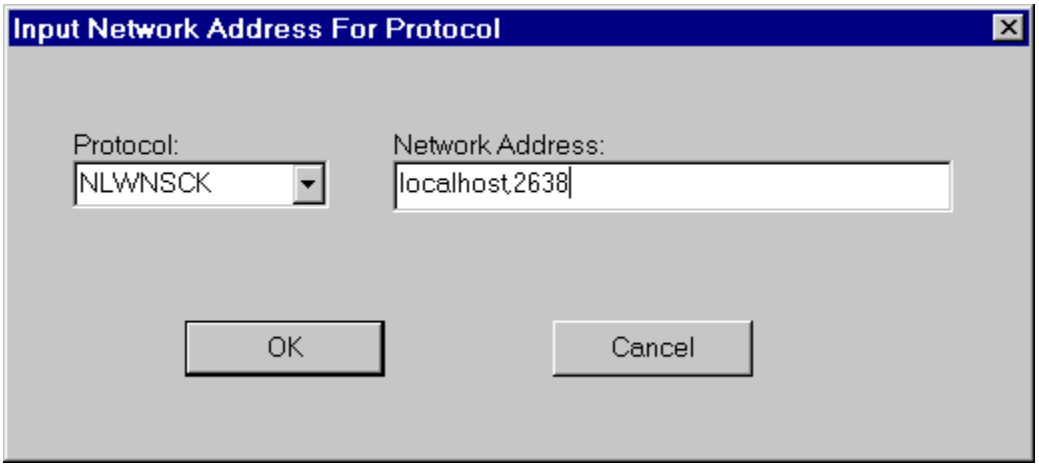
To enter a server address

1. In the **Server** box, select a server entry.
2. In the **Attributes** box, right-click the server address and choose **Modify Attribute**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

3. Click **Add**.

4. In the **Protocol** list, select **NLWNSCK** (this is the TCP/IP protocol).



5. In the **Network Address** field, type a valid network address. For TCP/IP addresses, use one of the following two forms:

   - computer name, port number
   - IP-address, portnumber

   The address or computer name is separated from the port number by a comma.

   Computer name A name (or an IP address) identifies the computer on which the server is running. On Windows operating systems, you can find the computer name in Network Settings, in the Control Panel.

   If your client and server are on the same computer, you must still enter the computer name. In this case, you can use **localhost** to identify the current computer.

   Port number The port number you enter must match the port number you used to start the SQL Anywhere database server. See "Starting the database server as an Open Server" on page 1167.

   The default port number for SQL Anywhere servers is 2638. This number has been assigned to SQL Anywhere by the Internet Adapter Number Authority (IANA), and use of this port is recommended unless you have good reasons for explicitly using another port.

   The following are valid server address entries:

```
elora,2638
123.85.234.029,2638
```

6. Click **OK**.

Verifying the server address

You can verify your network connection using the Ping command from the **Server Object** menu.

Database connections not verified
Verifying a network connection confirms that a server is receiving requests on the computer name and port number specified. It does not verify anything about database connections.

To ping a server

1. Ensure that the database server is running.
2. In the **Server** box of the **DSEdit** session window, click the server entry.
3. Choose **Server Object** » **Ping Server**.
4. Select the address you want to ping and click **Ping**.

   A window appears, notifying you whether the connection is successful. A window for a successful connection states that both open connection and close connection succeeded.

Renaming a server entry

You can rename server entries from the **DSEdit** session window.

To rename a server entry

1. In the **Server** box, select a server entry.
2. From the **Server Object** menu, choose **Rename**.
3. In the **Server Name** box, type a new name for the server entry.
4. Click **OK**.

Deleting server entries

You can delete server entries from the **DSEdit** session window.

To delete a server entry

1. In the **Server** box, select a server entry.
2. From the **Server Object** menu, choose **Delete**.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Configuring servers for JDBC

The JDBC connection address (URL) contains all the information required to locate the server. See "Supplying a URL to the driver" [*SQL Anywhere Server - Programming*].

Characteristics of Sybase Open Client and jConnect connections

When SQL Anywhere is serving applications over TDS, it automatically sets relevant database options to values compatible with Adaptive Server Enterprise default behavior. These options are set temporarily, for the duration of the connection only. The client application can override them at any time.

Default settings

The database options set on connection using TDS include:

Option	Set to
allow_nulls_by_default	Off
ansi_blanks	On
ansinull	Off
chained	Off
close_on_endtrans	Off
date_format	YYYY-MM-DD
date_order	MDY
escape_character	Off
isolation_level	1
on_tsql_error	Continue
quoted_identifier	Off
time_format	HH:NN:SS.SSS
timestamp_format	YYYY-MM-DD HH:NN:SS.SSS
tsql_variables	On

How the startup options are set

The default database options are set for TDS connections using a system procedure named sp_tsql_environment. This procedure sets the following options:

```
SET TEMPORARY OPTION allow_nulls_by_default='Off';
SET TEMPORARY OPTION ansi_blanks='On';
SET TEMPORARY OPTION ansinull='Off';
SET TEMPORARY OPTION chained='Off';
SET TEMPORARY OPTION close_on_endtrans='Off';
SET TEMPORARY OPTION date_format='YYYY-MM-DD';
SET TEMPORARY OPTION date_order='MDY';
SET TEMPORARY OPTION escape_character='Off';
SET TEMPORARY OPTION isolation_level='1';
SET TEMPORARY OPTION on_tsql_error='Continue';
SET TEMPORARY OPTION quoted_identifier='Off';
SET TEMPORARY OPTION time_format='HH:NN:SS.SSS';
SET TEMPORARY OPTION timestamp_format='YYYY-MM-DD HH:NN:SS.SSS';
SET TEMPORARY OPTION tsql_variables='On';
```

Do not edit the sp_tsql_environment procedure
Do not alter the sp_tsql_environment procedure yourself. It is for system use only.

The procedure sets options only for connections that use the TDS communications protocol. This includes Sybase Open Client and JDBC connections using jConnect. Other connections (ODBC and embedded SQL) have the default settings for the database.

You can change the options for TDS connections.

To change the option settings for TDS connections

1. Create a procedure that sets the database options you want. For example, you could use a procedure such as the following:

   ```
   CREATE PROCEDURE my_startup_procedure()
   BEGIN
     IF CONNECTION_PROPERTY('CommProtocol')='TDS' THEN
       SET TEMPORARY OPTION quoted_identifier='Off';
     END IF
   END;
   ```

   This particular procedure example changes only the quoted_identifier option from the default setting.

2. Set the login_procedure option to the name of a new procedure:

   ```
   SET OPTION login_procedure= 'DBA.my_startup_procedure';
   ```

Future connections will use the procedure. You can configure the procedure differently for different user IDs.

For more information about database options, see “Database options” on page 493.

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Index

Symbols

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

A

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

B

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

C

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

D

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

E

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

F

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

G

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

H

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

I

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

J

K

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

L

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

M

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

N

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

O

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

P

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

Q

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

R

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

S

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

T

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

U

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

V

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

W

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0

X

Z

 Copyright © 2010, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.0