SYBASE

iAnywhere.

SQL Anywhere® Server
Database Administration

Published: October 2006

Copyright and trademarks

Contents

About This Manual

Subject

This book covers material related to running, managing, and configuring SQL Anywhere databases. It describes database connections, the database server, database files, security, backup procedures, security, high availability, and replication with Replication Server, as well as administration utilities and options.

Audience

This manual is for all users of SQL Anywhere. It is to be used in conjunction with other manuals in the documentation set.

SQL Anywhere documentation

This book is part of the SQL Anywhere documentation set. This section describes the books in the documentation set and how you can use them.

The SQL Anywhere documentation

The complete SQL Anywhere documentation is available in two forms: an online form that combines all books, and as separate PDF files for each book. Both forms of the documentation contain identical information and consist of the following books:

- ♦ **SQL Anywhere 10 - Introduction** This book introduces SQL Anywhere 10—a comprehensive package that provides data management and data exchange, enabling the rapid development of database-powered applications for server, desktop, mobile, and remote office environments.
- ♦ **SQL Anywhere 10 - Changes and Upgrading** This book describes new features in SQL Anywhere 10 and in previous versions of the software.
- ♦ **SQL Anywhere Server - Database Administration** This book covers material related to running, managing, and configuring SQL Anywhere databases. It describes database connections, the database server, database files, security, backup procedures, security, and replication with Replication Server, as well as administration utilities and options.
- ♦ **SQL Anywhere Server - SQL Usage** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- ♦ **SQL Anywhere Server - SQL Reference** This book provides a complete reference for the SQL language used by SQL Anywhere. It also describes the SQL Anywhere system views and procedures.
- ♦ **SQL Anywhere Server - Programming** This book describes how to build and deploy database applications using the C, C++, and Java programming languages, as well as Visual Studio .NET. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by those tools.
- ♦ **SQL Anywhere 10 - Error Messages** This book provides a complete listing of SQL Anywhere error messages together with diagnostic information.
- ♦ **MobiLink - Getting Started** This manual introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.
- ♦ **MobiLink - Server Administration** This manual describes how to set up and administer MobiLink applications.
- ♦ **MobiLink - Client Administration** This manual describes how to set up, configure, and synchronize MobiLink clients. MobiLink clients can be SQL Anywhere or UltraLite databases.
- ♦ **MobiLink - Server-Initiated Synchronization** This manual describes MobiLink server-initiated synchronization, a feature of MobiLink that allows you to initiate synchronization or other remote actions from the consolidated database.

- **QAnywhere** This manual describes QAnywhere, which defines a messaging platform for mobile and wireless clients as well as traditional desktop and laptop clients.
- **SQL Remote** This book describes the SQL Remote data replication system for mobile computing, which enables sharing of data between a SQL Anywhere consolidated database and many SQL Anywhere remote databases using an indirect link such as email or file transfer.
- **SQL Anywhere 10 - Context-Sensitive Help** This manual provides context-sensitive help for the Connect dialog, the Query Editor, the MobiLink Monitor, the SQL Anywhere Console utility, the Index Consultant, and Interactive SQL.
- **UltraLite - Database Management and Reference** This manual introduces the UltraLite database system for small devices.
- **UltraLite - AppForge Programming** This manual describes UltraLite for AppForge. With UltraLite for AppForge you can develop and deploy database applications to handheld, mobile, or embedded devices, running Palm OS, Symbian OS, or Windows CE.
- **UltraLite - .NET Programming** This manual describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.
- **UltraLite - M-Business Anywhere Programming** This manual describes UltraLite for M-Business Anywhere. With UltraLite for M-Business Anywhere you can develop and deploy web-based database applications to handheld, mobile, or embedded devices, running Palm OS, Windows CE, or Windows XP.
- **UltraLite - C and C++ Programming** This manual describes UltraLite C and C++ programming interfaces. With UltraLite you can develop and deploy database applications to handheld, mobile, or embedded devices.

Documentation formats

SQL Anywhere provides documentation in the following formats:

- **Online documentation** The online documentation contains the complete SQL Anywhere documentation, including the books and the context-sensitive help for SQL Anywhere tools. The online documentation is updated with each maintenance release of the product, and is the most complete and up-to-date source of documentation.

  To access the online documentation on Windows operating systems, choose Start ► Programs ► SQL Anywhere 10 ► Online Books. You can navigate the online documentation using the HTML Help table of contents, index, and search facility in the left pane, as well as using the links and menus in the right pane.

  To access the online documentation on Unix operating systems, see the HTML documentation under your SQL Anywhere installation or on your installation CD.

- **PDF files** The complete set of SQL Anywhere books is provided as a set of Adobe Portable Document Format (pdf) files, viewable with Adobe Reader.

  On Windows, the PDF books are accessible from the online books via the PDF link at the top of each page, or from the Windows Start menu (Start ► Programs ► SQL Anywhere 10 ► Online Books - PDF Format).

On Unix, the PDF books are accessible on your installation CD.

Documentation conventions

This section lists the typographic and graphical conventions used in this documentation.

Syntax conventions

The following conventions are used in the SQL syntax descriptions:

- **Keywords** All SQL keywords appear in uppercase, like the words ALTER TABLE in the following example:

  ALTER TABLE [*owner*.]*table-name*

- **Placeholders** Items that must be replaced with appropriate identifiers or expressions are shown like the words *owner* and *table-name* in the following example:

  ALTER TABLE [*owner*.]*table-name*

- **Repeating items** Lists of repeating items are shown with an element of the list followed by an ellipsis (three dots), like *column-constraint* in the following example:

  ADD *column-definition* [*column-constraint*, …]

  One or more list elements are allowed. In this example, if more than one is specified, they must be separated by commas.

- **Optional portions** Optional portions of a statement are enclosed by square brackets.

  RELEASE SAVEPOINT [*savepoint-name*]

  These square brackets indicate that the *savepoint-name* is optional. The square brackets should not be typed.

- **Options** When none or only one of a list of items can be chosen, vertical bars separate the items and the list is enclosed in square brackets.

  [**ASC** | **DESC**]

  For example, you can choose one of ASC, DESC, or neither. The square brackets should not be typed.

- **Alternatives** When precisely one of the options must be chosen, the alternatives are enclosed in curly braces and a bar is used to separate the options.

  [**QUOTES** { **ON** | **OFF** }]

  If the QUOTES option is used, one of ON or OFF must be provided. The brackets and braces should not be typed.

File name conventions

The documentation generally adopts Windows conventions when describing operating-system dependent tasks and features such as paths and file names. In most cases, there is a simple transformation to the syntax used on other operating systems.

- **Directories and path names** The documentation typically lists directory paths using Windows conventions, including colons for drives and backslashes as a directory separator. For example,

  ```
  MobiLink\redirector
  ```

  On Unix, Linux, and Mac OS X, you should use forward slashes instead. For example,

  ```
  MobiLink/redirector
  ```

- **Executable files** The documentation shows executable file names using Windows conventions, with the suffix *.exe*. On Unix, Linux, and Mac OS X, executable file names have no suffix. On NetWare, executable file names use the suffix *.nlm*.

  For example, on Windows, the network database server is *dbsrv10.exe*. On Unix, Linux, and Mac OS X, it is *dbsrv10*. On NetWare, it is *dbsrv10.nlm*.

- **install-dir** The installation process allows you to choose where to install SQL Anywhere, and the documentation refers to this location using the convention *install-dir*.

  After installation is complete, the environment variable SQLANY10 specifies the location of the installation directory containing the SQL Anywhere components (*install-dir*). SQLANYSH10 specifies the location of the directory containing components shared by SQL Anywhere with other Sybase applications.

  For more information on the default location of *install-dir*, by operating system, see "File Locations and Installation Settings" on page 293.

- **samples-dir** The installation process allows you to choose where to install the samples that are included with SQL Anywhere, and the documentation refers to this location using the convention *samples-dir*.

  After installation is complete, the environment variable SQLANYSAMP10 specifies the location of the directory containing the samples (*samples-dir*). From the Windows Start menu, choosing Programs ▶ SQL Anywhere 10 ▶ Sample Applications and Projects opens a Windows Explorer window in this directory.

  For more information on the default location of *samples-dir*, by operating system, see "The samples directory" on page 295.

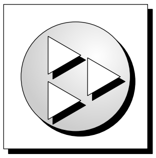
- **Environment variables** The documentation refers to setting environment variables. On Windows, environment variables are referred to using the syntax *%envvar%*. On Unix, Linux, and Mac OS X, environment variables are referred to using the syntax *$envvar* or *${envvar}*.

Unix, Linux, and Mac OS X environment variables are stored in shell and login startup files, such as *.cshrc* or *.tcshrc*.

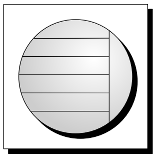
Graphic icons

The following icons are used in this documentation.

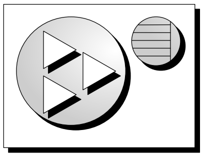
- A client application.



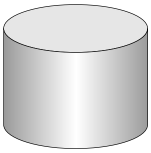
- A database server, such as SQL Anywhere.



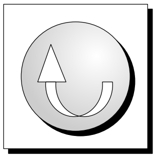
- An UltraLite application.



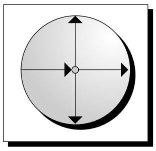
- A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



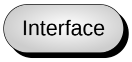
- Replication or synchronization middleware. These assist in sharing data among databases. Examples are the MobiLink server and the SQL Remote Message Agent.



- A Sybase Replication Server



- A programming interface.



Finding out more and providing feedback

Finding out more

Additional information and resources, including a code exchange, are available at the iAnywhere Developer Network at http://www.ianywhere.com/developer/.

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

When you write to one of these newsgroups, always provide detailed information about your problem, including the build number of your version of SQL Anywhere. You can find this information by entering **dbeng10 -v** at a command prompt.

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

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

Feedback

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

You can email comments and suggestions to the SQL Anywhere documentation team at iasdoc@ianywhere.com. Although we do not reply to emails sent to that address, we read all suggestions with interest.

In addition, you can provide feedback on the documentation and the software through the newsgroups listed above.

Part I. 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.

CHAPTER 1

Tutorial: Using the Sample Database

Contents

About this chapter

This chapter provides basic information about the database servers and the sample database. It shows you how to make a copy of the sample database, how to start the database server running the sample database, how to view the Server Messages window, and how to stop a database server.

Lesson 1: Make a copy of 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), as well as product information (products) and sales information (sales orders, customers, and contacts). All information in the sample database is fictional.

☞ For more information, see "About the sample database" [*SQL Anywhere 10 - Introduction*].

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. Copy the sample database from *samples-dir\demo.db* to *c:\demodb*.

   ☞ For information about the default location of *samples-dir*, see "The samples directory" on page 295.

Lesson 2: Start the SQL Anywhere database server

In this section, you start the SQL Anywhere personal database server running the sample database.

☞ For information about starting the network database server, see "Connecting to a server on a network" on page 60.

♦ To start a personal database server running the sample database (Windows)

- From the Start menu, choose Programs ► SQL Anywhere 10 ► SQL Anywhere ► Personal Server Sample.

  This starts a personal database server running the sample database. The Server Messages window appears briefly, then disappears. The database server then appears as an icon in the system tray.

♦ To start a personal database server running the sample database (Command prompt)

1. Open a command prompt.
2. Change to the SQL Anywhere installation directory.
3. Start the personal database server running the sample database.

   Enter the following command to start the personal database server, assign the server name demo10 using the -n server option, and connect to the sample database:

   ```
   dbeng10 -n demo10 c:\demodb\demo.db
   ```

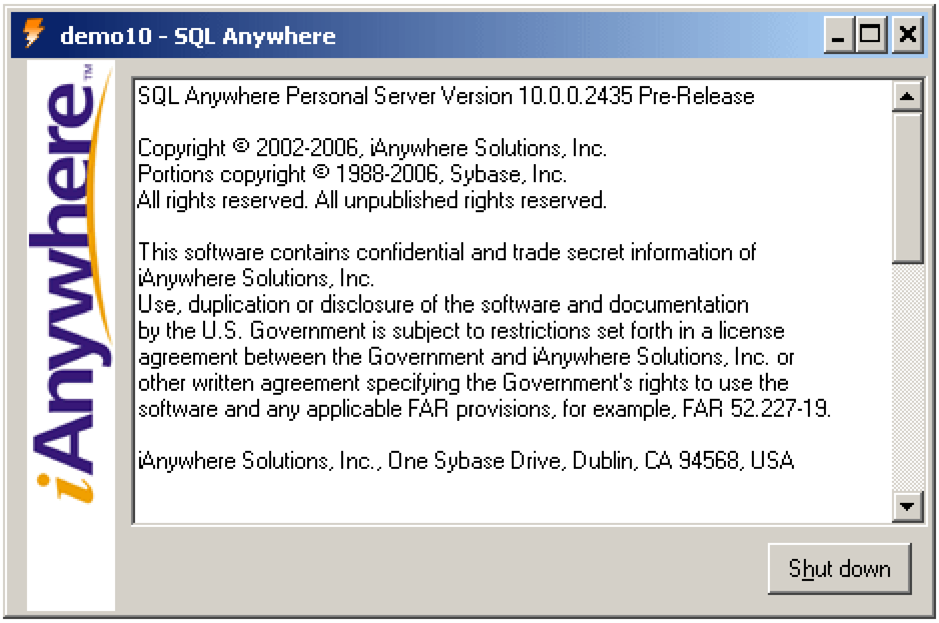
   On Windows, the database server appears as an icon in the system tray.

☞ For more information, see "Running the Database Server" on page 11.

Lesson 3: Display the 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 Server Messages window in Windows by double-clicking the SQL Anywhere personal server icon in the system tray.



```
SQL Anywhere Personal Server Version 10.0.0.2435 Pre-Release

Copyright © 2002-2006, iAnywhere Solutions, Inc.
Portions copyright © 1988-2006, Sybase, Inc.
All rights reserved. All unpublished rights reserved.

This software contains confidential and trade secret information of
iAnywhere Solutions, Inc.
Use, duplication or disclosure of the software and documentation
by the U.S. Government is subject to restrictions set forth in a license
agreement between the Government and iAnywhere Solutions, Inc. or
other written agreement specifying the Government's rights to use the
software and any applicable FAR provisions, for example, FAR 52.227-19.

iAnywhere Solutions, Inc., One Sybase Drive, Dublin, CA 94568, USA
```

The Server Messages window displays useful information, including the following:

- **The server name** The name in the title bar (in this case **demo10**) 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 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 server and the databases" on page 18.
- **The version and build numbers** The numbers following the server name (for example, **10.0.0.2435**) 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 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.

☞ For more information, see “The Database Server” on page 117.

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 Server Messages window.

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

1. Double-click the SQL Anywhere icon in the Windows taskbar.

   The Server Messages window appears.

2. Click Shut Down.

♦ To stop the database server running the sample database (Command prompt)

1. Open a command prompt.
2. Change to the SQL Anywhere installation directory.
3. Stop the database server running the sample database.

   Enter the following command to shut down the personal database server:

   ```
   dbstop demo10
   ```

4. Delete the *c:\demodb* directory.

The Stop Server utility (dbstop) can only be run at a command prompt and is not available on NetWare. See "The Stop Server utility" on page 682.

Summary

In this tutorial, you learned how to make a copy of the sample database, how to start a database server running the sample database, and how to view the contents of the Server Messages window. You also learned how to stop the database server.

See also

- "Starting Interactive SQL" on page 540
- "Connecting from Sybase Central, Interactive SQL, or the SQL Anywhere Console utility" on page 54

CHAPTER 2

Running the Database Server

Contents

About this chapter

This chapter describes how to start and stop the SQL Anywhere database server, and the options available to you on startup under different operating systems.

Introduction

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 CE, the name of the personal server executable is *dbeng10.exe*. On Unix operating systems its name is dbeng10. Only the network server is supported on Windows CE and NetWare.

- **The network database server** Intended for multi-user use, this executable supports client/server communications across a network.

  On Windows operating systems, including Windows CE, the name of the network server executable is *dbsrv10.exe*. On Novell NetWare the name is *dbsrv10.nlm* and on Linux and Unix operating systems it is dbsrv10.

Server differences

The request-processing engine is identical in both the personal and network servers. Each one supports exactly the same SQL, and exactly the same database features. The main differences include:

- **Network protocol support** Only the network server supports communications across a network.
- **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 651.
- **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. In addition, the personal database server and runtime database server are both limited to a single processor.
- **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, NetWare, Linux, and Unix operating systems.

SQL Anywhere supports the TCP/IP network protocol. The SPX protocol is also supported for Novell NetWare.

First steps

You can start a personal server running a single database in several ways:

♦ On Windows, from the Start menu, choose Programs ▶ SQL Anywhere 10 ▶ SQL Anywhere ▶ Personal Server Sample.

♦ Execute the following command in the directory where *demo.db* is located to start both a personal server and a database called *demo.db*:

```
dbeng10 demo
```

♦ Use a database file name in a connection string.

For more information, see "Connecting to an embedded database" on page 58.

Where to specify commands

You can specify commands in several ways, depending on your operating system. For example, you can:

♦ enter 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.

For more information, see "StartLine connection parameter [START]" on page 238.

There are slight variations in how you specify the basic command from platform to platform. These are described in the following section.

Start the database server

The way you start the database server varies slightly depending on the operating system you use. This section describes how to specify commands for the simple case of running a single database with default settings on each supported operating system.

Notes

♦ Except where otherwise noted, these commands start the personal server (**dbeng10**). To start a network server, replace **dbeng10** with **dbsrv10**.

♦ 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.

♦ To start the personal database server using default options (Windows except Windows CE)

1. Open a command prompt.

2. Enter the following command:

   ```
   dbeng10 path\database-file
   ```

   If you omit the database file, the Server Startup Options dialog appears where you can locate a database file using Browse.

☞ For information about starting a database server on Windows CE, see "Connecting to a database running on a Windows CE device" on page 971.

♦ To start the personal database server using default options (Unix)

1. Open a command prompt.
2. Enter the following command:

   ```
   dbeng10 path/database-file
   ```

There is no personal server for Novell NetWare, just a network server. The following command starts the network server on NetWare.

♦ To start the database server using default options (NetWare)

- The database server for NetWare is a NetWare Loadable Module (NLM) (*dbsrv10.nlm*). An NLM is a program that you can run on your NetWare server. Load a database server on your NetWare server as follows:

  ```
  load dbsrv10.nlm path\database-file
  ```

  The database file must be on a NetWare volume. A typical file name has the form *DB:\database\sales.db*.

  You can load the server from a client computer using the Novell remote console utility. See your NetWare documentation for details.

  You can put the command into your NetWare *autoexec.ncf* file so that SQL Anywhere loads automatically each time you start the NetWare server.

What else is there to it?

Although you can start a personal server in the simple way described in the previous section, there are many other aspects to running a database server in a production environment. For example,

♦ 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.

♦ You can run the server as a Windows **service**. This allows it to continue running even when you log off the computer.

- You can start the personal server from an application and shut it down when the application has finished with it. This is typical when using the database server as an **embedded database**.

The remainder of this chapter describes these options in more detail.

Starting the database server

The general form for the server command is as follows:

executable [*server-options*] [*database-file* [*database-options*], …]

If you supply no options and no database file, then on Windows operating systems a dialog appears, allowing you to Browse to your database file.

The elements of the database server command include the following:

♦ **Executable** This can be either the personal server or the network server.

For the executable names on different operating systems, see "Introduction" on page 12.

In this chapter, unless discussing network-specific options, the personal server is used in sample commands. The network server takes a very similar set of options.

♦ **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. Database files and transaction log files located on a network drive can lead to poor performance, data corruption, and server instability.
> For best results, the transaction log should be kept on a different disk from the database files. See "The transaction log" on page 766.

♦ **Database options** For each database file you start, you can provide database options that control certain aspects of its behavior.

For full reference information on each of these options, see "The SQL Anywhere database server" on page 118.

In examples throughout this chapter where there are several options, they appear on separate lines for clarity, as they could be written in a configuration file. If you enter them directly at the command prompt, you must enter them all on one line.

Case sensitivity

Database and server options are generally case sensitive. You should enter all options in lowercase.

Listing available options

♦ **To list the database server options**

1. Open a command prompt.

2. Enter the following command:

```
dbeng10 -?
```

Logging database server actions

Logging the actions that the server takes is particularly useful during the development process and when troubleshooting.

Logging output to a file

Logging output is sent to the Server Messages window. In addition, you can send the output to a log file using the -o option. The following command sends output to a log file named *dbsrv.log*.

```
dbsrv10 -o dbsrvsrv.log -c ...
```

You can control the size of console log files, and specify what you want done when a file reaches its maximum size.

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

☞ For more information, see:

- "-o server option" on page 161
- "-on server option" on page 161
- "-os server option" on page 162
- "-ot server option" on page 163

You can specify a separate file where startup errors, fatal errors, and assertions are logged using the -oe option. See "-oe server option" on page 161.

Some common options

This section describes some of the most common options, and points out when you may want to use them. They are:

- Using configuration files
- Naming the server and the databases
- Performance
- Permissions
- Maximum page size
- Special modes
- Threading
- Network communications (network server only)

Using configuration files to store server startup options

If you use an extensive set of options, you can store them in a configuration file and invoke that file in a server command. The configuration file can contain options on several lines. For example, the following configuration file starts the 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, you would use a forward slash instead of the backslash in the file path.

For information about the default location of *samples-dir*, see "The samples directory" on page 295.

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

```
dbeng10 @sample.cfg
```

☞ For more information, see "@data server option" on page 125 and "Using configuration files" on page 587.

Naming the server and the databases

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

The server and database names are among the connection parameters that client applications may use when connecting to a database. The server name appears on the desktop icon and in the title bar of the Server Messages window.

Naming the server

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

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

```
dbeng10 -n Cambridge samples-dir\demo.db
```

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

```
dbeng10 -n Galt
```

The maximum length of the server name depends on the protocol being used:

Protocol	Truncation length
TCP/IP	250 bytes
Shared memory	250 bytes
SPX	32 bytes

☞ For more information about starting databases on a running server, see "Starting and stopping databases" on page 32.

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.

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.

```
dbeng10 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:

```
dbeng10 samples-dir\demo.db -n MyDB
```

Case sensitivity

Server names and database names are case insensitive as long as the character set is single-byte.

☞ For more information, see “Connection strings and character sets” on page 322.

Controlling performance and memory from the command line

Several options can have a major impact on database server performance, including:

- **Cache size** The -c option controls the amount of memory that SQL Anywhere uses as a cache. This can be a major 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 many times 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. On Windows and Unix operating systems, the database server automatically grows the cache when the available cache is exhausted.

  The 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( 'CacheSize' )
  ```

  For more information about performance tuning, see “Monitoring and Improving Performance” [*SQL Anywhere Server - SQL Usage*].

  For information on controlling cache size, see “-c server option” on page 127.

- **Multiprogramming level** The database server's multiprogramming level is the maximum number of server tasks that can execute concurrently. In general, a higher multiprogramming level increases the overall throughput of the 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 actually increase transaction response time, or in some cases even lower the system's throughput. You can set the server's multiprogramming level with the -gn option.

  For more information, see “-gn server option” on page 149 and “Setting the database server's multiprogramming level” on page 25.

- **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.

  For more information, see “-gt server option” on page 152 and “Threading in SQL Anywhere” on page 22.

- **Other performance-related options** There are several options available for tuning network performance, including -gb (database process priority), and -u (buffered disk I/O).

For more information about startup options, see “The SQL Anywhere database server” on page 118.

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.

☞ For more information, see “Running the database server in a secure fashion” on page 872.

Setting a maximum page size

The database server cache is arranged in **pages**—fixed-size areas of memory. Since the 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 on the command line. Every database page must fit into a cache page. By default, the server page size is the same as the largest page size of the databases on the command line. Once the server starts, you cannot start a database with a larger page size than the server.

To allow databases with larger page sizes to be started after startup, you can force the server to start with a specified page size 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 server that reserves a 64 MB cache and can accommodate databases of page sizes up to 8192 bytes.

```
dbsrv10 -gp 8192 -c 64M -n myserver
```

Running in special modes

You can run SQL Anywhere in special modes for particular purposes.

- **Read-only** You can run databases in read-only mode by supplying the -r option. You cannot start databases with auditing turned on in read-only mode.

  For more information, see “-r server option” on page 167.

- **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.

  For more information, see “-b server option” on page 127, 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.

For more information, see "The SQL Anywhere database server" on page 118.

Threading in SQL Anywhere

To understand the SQL Anywhere threading model, you must also 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, 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 cancelled.

♦ **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 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 housekeeping chores, such as running the cleaner or processing timers. The maximum number of active tasks that can execute concurrently is set by the -gn option. If more tasks arrive at the database server than can be concurrently processed, they are queued for execution. If an active task, or a task for which processing has already begun, needs to block for some reason during its processing, such as while waiting for a lock, or for I/O to complete, it is still considered to be active. It therefore counts against the upper bound place by the value of the -gn option.

♦ **Thread** A **thread** is an operating system construct that represents an executing **thread-of-control** within an application. Every operating system process, including the database server, is executed by at least one, and possibly many threads. A thread is scheduled outside of the application by the operating system, and ultimately, all of an application's execution is performed by its threads. Tasks, within a SQL Anywhere database server, execute on an operating system thread. At startup, SQL Anywhere creates a fixed number of threads, controlled by the -gtc option (on Windows and Linux), or the -gn option (on Unix or NetWare).

☞ For a list of options that control threading, see "Controlling threading behavior" on page 23.

See also

♦ "Parallelism during query execution" [*SQL Anywhere Server - SQL Usage*]
♦ "-gn server option" on page 149
♦ "-gtc server option" on page 153
♦ "sa_clean_database system procedure" [*SQL Anywhere Server - SQL Reference*]
♦ "Transaction blocking and deadlock" [*SQL Anywhere Server - SQL Usage*]

Tasks on Unix and NetWare

On Unix and NetWare, a task is executed directly on an operating system thread. On these platforms, the value of the -gn option sets the number of operating system threads created when the database server starts; all tasks are serviced from this set of threads. When a thread becomes available, it picks up the next available task that requires processing. Once processing a task, a thread remains with that task until it has been completed. If the task needs to block for some reason, perhaps because it is pending an I/O operation, or while waiting for a lock, the thread voluntarily relinquishes control of the CPU back to the operating system scheduler allowing other threads to run on that CPU.

In addition to voluntarily relinquishing the CPU, a thread may be preempted by the operating system scheduler. Each application thread within a process is given a series of time slices in which to run, the length of which is determined by its priority and other system factors. When a thread reaches the end of its current time slice it is preempted by the operating system and scheduled to run again at a later time. The operating system scheduler then chooses another thread to execute for a time slice. 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.

Once processing of the active task is completed, the thread checks to see if any other tasks have come available for processing. If so, it picks up the next available task and continues. Otherwise, it relinquishes the CPU and waits for a new task to arrive at the database server.

See also

- “-gn server option” on page 149

Tasks on Windows XP/200x and Linux

On Windows XP/200x and Linux, tasks are executed on lightweight threads known as **fibers**. Fibers allow tasks running on threads to schedule amongst themselves co-operatively, rather than relying on the operating system thread scheduler. The result is that a context switch between fibers is much less expensive than a thread context switch as there is no interaction with the operating system kernel or scheduler. In multi-threaded applications that otherwise do frequent thread context switching, the use of fibers can dramatically improve performance and scalability.

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 will relinquish 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 platforms that support fibers, there are at least as many fibers created as required by the maximum concurrency setting of the server, as specified by the -gn option. The server may create more than this value so there is always a fiber available to service internal server tasks. See “-gn server option” on page 149.

Controlling threading behavior

There are five main 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 server's multiprogramming level. This value determines the maximum number of tasks that may be active at one time. Each database request uses at least one task, and possibly more if intra-query parallelism is involved. Additionally, the server will occasionally schedule tasks to perform internal housekeeping activities. When the number of tasks in the server exceeds the multiprogramming level, the outstanding tasks must wait until a currently-running, or active task completes. By default, a maximum of 20 tasks can execute concurrently for the network database server and the personal database server. See "-gn server option" on page 149, and "Setting the database server's multiprogramming level" on page 25.

- **Stack size per internal execution thread (-gss server option)**
  You can set the stack size per internal execution thread in the server using the -gss option. The -gss option allows lowering the memory usage of the database server, which may be useful in environments with limited memory. This option has no effect on Windows operating systems. See "-gss server option" on page 151.

- **Number of processors (-gt server option)**
  If you have more than one processor, you can control how many processors the threads exploit by specifying the -gt option. This option is not supported on NetWare. See "-gt server option" on page 152.

- **Number of threads assigned to the database server process (-gx server option)**
  On Windows XP/200x, the -gx option controls the number of threads that are dedicated to hosting fibers to service requests. By default, this is set to one more than the number of CPUs on the computer. On Unix and NetWare, where each task is its own thread, this option is not needed.

- **Processor concurrency (-gtc server option)**
  You can specify the maximum number of threads that can run concurrently on a CPU. By default, the database server runs on all hyperthreads and cores of each licensed physical processors. This option is not supported on NetWare. See "-gtc server option" on page 153.

Threading tips

- Increasing -gn can reduce the chance of thread deadlock occurring. See "-gn server option" on page 149.

- Setting -gt to 1 can help work around concurrency problems. See "-gt server option" on page 152.

- Investigating the Performance Monitor readings for Requests: Active and Requests: Unscheduled can help you determine an appropriate value for -gn on Windows XP/200x. If the number of active requests is always less than -gn, you can lower -gn. If the number of total requests (active + unscheduled) is often larger than -gn, then you might want to increase the value for -gn. See "Performance Monitor statistics" [*SQL Anywhere Server - SQL Usage*], and "-gn server option" on page 149.

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 purpose of 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 ♦ -gn 20 ♦ -gx 21 Threads can execute on A0 and A1.
No licensing restrictions on the CPU with -gtc 5 specified	♦ -gt 4 ♦ -gtc 5 ♦ -gn 20 ♦ -gx 21 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 ♦ -gn 20 ♦ -gx 21 Threads can execute on A0, A1, B0, B1, and C0.
No licensing restrictions on the CPU with -gtc 1 specified	♦ -gt 4 ♦ -gtc 1 ♦ -gn 20 ♦ -gx 2 Threads can execute only on A0.

Setting 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, and is controlled through the specification of the -gn server option. An active task is one that is currently being executed by a thread (or fiber) in the database server. An active task may be executing an access plan operator, or performing some other useful work, but may also be blocked, waiting for a resource (such as an I/O operation, or a lock on a row). An unscheduled task is one that is ready to execute, but is waiting for an available thread or fiber. 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 on the computer.

The multiprogramming level remains constant during server execution, and applies to all databases on that server. The default is 20 active tasks for the network database server and for the personal database server, except on Windows CE where the default is 3.

Raising the multiprogramming level

It can be difficult to determine when to raise or lower the multiprogramming level. For example, if a database application makes use of Java stored procedures, or if intra-query parallelism is enabled, then the additional server tasks created to process these requests may exceed the multiprogramming limit, and execution of these tasks will wait until another request completes. In this case, raising the multiprogramming level may be appropriate. In many cases, increases to the multiprogramming level will correspondingly increase the database server's overall throughput, as doing so permits additional tasks (requests) 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 thread (in the case of Windows and Linux, a lightweight thread called a fiber) and additional bookkeeping structures to control its scheduling. In addition, each active task requires the preallocation of address space for its execution stack. The size of the stack varies by platform, but is roughly 1 MB on 32-bit platforms, and larger on 64-bit platforms. On Windows systems, the allocation of stack space affects the address space of the server process, but the stack memory is allocated on demand. On Unix platforms, including Linux, the backing memory for the stack is allocated immediately. Hence, setting a higher multiprogramming level increases the server's memory footprint, and reduces the amount of memory available for the cache 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. This can occur, for example, when too many active requests are competing for space in the database cache, but the cache is insufficiently large to accommodate the working set of database pages used by the set of active requests. 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 must ensure adequate resources exist at all times to handle the maximal number of tasks as specified by the -gn option. The most critical resource is memory. Query execution operators such as hash join or sorting can require significant amounts of memory to execute efficiently.

A memory usage governor exists for each task that limits the amount of memory that should be used for this task. This soft limit is defined by the following formula:

```
(total size of the database cache) / (multiprogramming level)
```

If a task exceeds this threshold, the database kernel requests any query execution operators for that task to free unnecessary memory buffers (if possible) so that the that the limit is no longer exceeded. This can result in execution operators switching to low-memory execution strategies at run time, which conserve memory at the expense of slower execution times.

The query optimizer takes this memory threshold into account when costing access plans, and refrains from choosing execution strategies that rely on larger amounts of memory than this threshold. In addition, each task has a limit of the following:

```
(1/4 maximum cache size) / (number of currently active tasks)
```

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 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.

In SQL Anywhere, threads (fibers) execute tasks in a cooperative fashion. Once a task has completed, the thread (fiber) is free to pick up the next task awaiting execution. However, if a task is blocked, for example when waiting for row lock, the thread (fiber) is also blocked.

When the multiprogramming level is set too low, this can cause **thread deadlock**. Suppose that the database server has *n* threads (fibers). Thread deadlock occurs when *n*-1 threads are blocked, and the last thread is about to block. The database server's kernel cannot permit this last thread to block, since doing so would result in all threads being blocked, and the server would hang. Rather, the database server terminates the task that is about to block the last thread with SQLSTATE 40W06.

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 consequently impairs scalability. One example of this 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 consequence is that it effectively serializes all of 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.

Choosing the multiprogramming level

It is recommended that you experiment with your application's workload to analyze the effects of the server's multiprogramming level on server throughput and request response time. Various performance counters are available as either property functions, or through the Windows Performance Monitor on Windows XP/200x, to help you analyze database server behavior when testing your application. The performance counters related to active and unscheduled requests are particularly important to this analysis.

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 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 -gn, then an increase in the multiprogramming level may be warranted, subject to the tradeoffs outlined above. Note that the Performance Monitor is not available for NetWare, Unix, or Linux platforms.

Selecting communications protocols

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

By default, the database server starts all available protocols. You can limit the protocols available to a database server using the -x option. On the client side, many of the same options can be controlled using the CommLinks (LINKS) connection parameter.

☞ For more information on running the server using these options, see "Supported network protocols" on page 104.

Available protocols for the personal server

The personal database server (*dbeng10.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 for same-computer communications only from TDS clients, Open Client, or the jConnect JDBC driver. You must run TCP/IP if you want to connect from Open Client or jConnect.

  For more information on TDS clients, see "SQL Anywhere as an Open Server" on page 919.

Available protocols for the network server

The network database server (*dbsrv10.exe*) supports the following protocols:

- **Shared memory** This protocol is for same-computer communications, and always remains available. It is available on all platforms.
- **SPX** This protocol is supported on all platforms except for Unix.
- **TCP/IP** This protocol is supported 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.

Specifying protocols

By using 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 using the TCP/IP protocol:

```
dbsrv10 -x "tcpip" samples-dir\demo.db
```

Although not strictly required in this example, the quotes are necessary if there are spaces in any of the arguments to -x.

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, one with a specified port number.

```
dbsrv10 -x "tcpip(MyIP=192.75.209.12:2367,192.75.209.32)" samples-dir\demo.db
```

☞ For more information about available network protocol options that can serve as part of the -x option, see "Network protocol options" on page 242.

Stopping the database server

You can stop the database server by:

- Clicking Shutdown on the Server Messages window.
- Using the dbstop utility.

  The dbstop utility is particularly 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 682.
- Letting it shut down automatically by default when the application disconnects. (This only works if the server is a personal server started by an application connection string.)
- Pressing Q when the Server Messages window has the focus on Unix or NetWare computers.

Examples

♦ To stop a server using the dbstop utility

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

```
dbsrv10 -n Ottawa samples-dir\demo.db
```

2. Stop the server using dbstop:

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

Who can stop the server?

When you start a 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. (Interactively, of course, anybody at the computer can click Shut Down on the Server Messages window.)

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 761.

It is better to stop the database server explicitly before closing the operating system session. On NetWare, however, shutting down the NetWare server computer properly does stop the database server cleanly.

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 server at the same time, as follows:

```
dbeng10 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.

  For more information, see "Connecting to an embedded database" on page 58, or "DatabaseFile connection parameter [DBF]" on page 216.

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

  For more information, 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 21.
- The -gd server option decides the permissions required to start databases.

Stopping 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.

  For more information, see "AutoStop connection parameter [ASTOP]" on page 209.

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

  For more information, see "STOP DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

Running the 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 terminate.

It is common to require database servers to be available all the time. To make this easier, you can run SQL Anywhere for Windows XP/200x and for Unix in such a way that, when you log off the computer, the database server remains running. The way you do this depends on your operating system.

- ♦ **Windows service** You can run the Windows database server as a service. This has many convenient properties for running high availability servers. See "Understanding Windows services" on page 34.
- ♦ **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 33.
- ♦ **Linux service** You can run the Linux database server as a service. This has many convenient properties for running high availability servers. See "Service utility (dbsvc) for Linux" on page 660.

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 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.

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:

   ```
   dbsrv10 -ud demo
   ```

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

   ```
   dbspawn dbsrv10 demo
   ```

   One advantage of using dbspawn is that the dbspawn process does not terminate 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 dbeng10 and dbsrv10 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 dbsrv10 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 dbsrv10 demo
if [ $? != 0 ]; then
   echo Failed to start demo server
   exit
fi
# ok, now you can start the applications
...
```

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

```
...
if( fork() == 0 ) {
      /* child process = start server daemon */
      execl( "/opt/sqlanywhere10/bin/dbsrv10",
"dbsrv10", "-ud", "demo" );
   exit(1);
}
/* parent process */
...
```

Note that the -ud option is used.

☞ For more information, see "-ud server option" on page 179 and "The Start Server in Background utility" on page 680.

Understanding Windows services

Although you can run the database server like any other Windows XP/200x program rather than as a service, there are limitations to running it as a standard program, particularly in multi-user environments.

Limitations of running as a standard executable

When you start a program, it runs under your Windows XP/200x login session, which means that if you log off the computer, the program terminates. This 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 can also present a security risk as the Windows XP/200x 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 "The Service utility" on page 654.

Programs that can be run as Windows services

You can run the following programs as services:

- Network database server (*dbsrv10.exe*)
- Personal database server (*dbeng10.exe*)
- SQL Anywhere Broadcast Repeater utility (*dbns10.exe*)
- MobiLink server (*mlsrv10.exe*)
- SQL Remote Message Agent (*dbremote.exe*)
- Log Transfer Manager utility (*dbltm.exe*)
- Listener utility (*dblsn.exe*)
- MobiLink synchronization client (*dbmlsync.exe*)

Note that not all of these applications are supplied in all editions of SQL Anywhere.

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.

Creating a Windows service

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

♦ To create a new service (Sybase Central)

1. In the left pane, select the SQL Anywhere 10 plug-in.
2. In the right pane, click the Services tab.
3. From the File menu, choose New ▶ Service.

   The Create Service wizard appears.
4. Follow the instructions in the wizard.

♦ To create a new service (Command line)

1. Open a command prompt.
2. Execute the Service Creation utility using the -w option.

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

   ```
   dbsvc -as -w myserv "c:\Program Files\SQL Anywhere 10\win32\dbeng10.exe"
   -n william -c 8m "c:\temp\sample.db"
   ```

☞ For more information about the Service Creation utility and options, see "The Service utility" on page 654.

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 box of the Create Service wizard in Sybase Central:

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

  Line breaks are optional.

  For information on valid options, see the description of each program in "Database Administration Utilities" on page 585.
- 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 39.

- If you want the service to be accessible from the Windows desktop, check 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.

☞ For more information on the configuration options, see "Configuring Windows services" on page 37.

Deleting a Windows service

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

If you want to re-install a service you previously deleting, you need to re-type the options.

♦ To deleting a Windows service (Sybase Central)

1. From the Context dropdown list, choose the SQL Anywhere 10 plug-in.

   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)

1. Open a command prompt.

2. Execute the Service utility using the -d option.

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

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

☞ For more information about the Service utility and options, see "The Service utility" on page 654.

Configuring Windows services

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

☞ For a full description of the options for each of the administration utilities, see "Database Administration Utilities" on page 585.

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

1. From the Context dropdown list, choose the SQL Anywhere 10 plug-in.

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 property sheet.
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 startup options

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

- **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

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

Examples

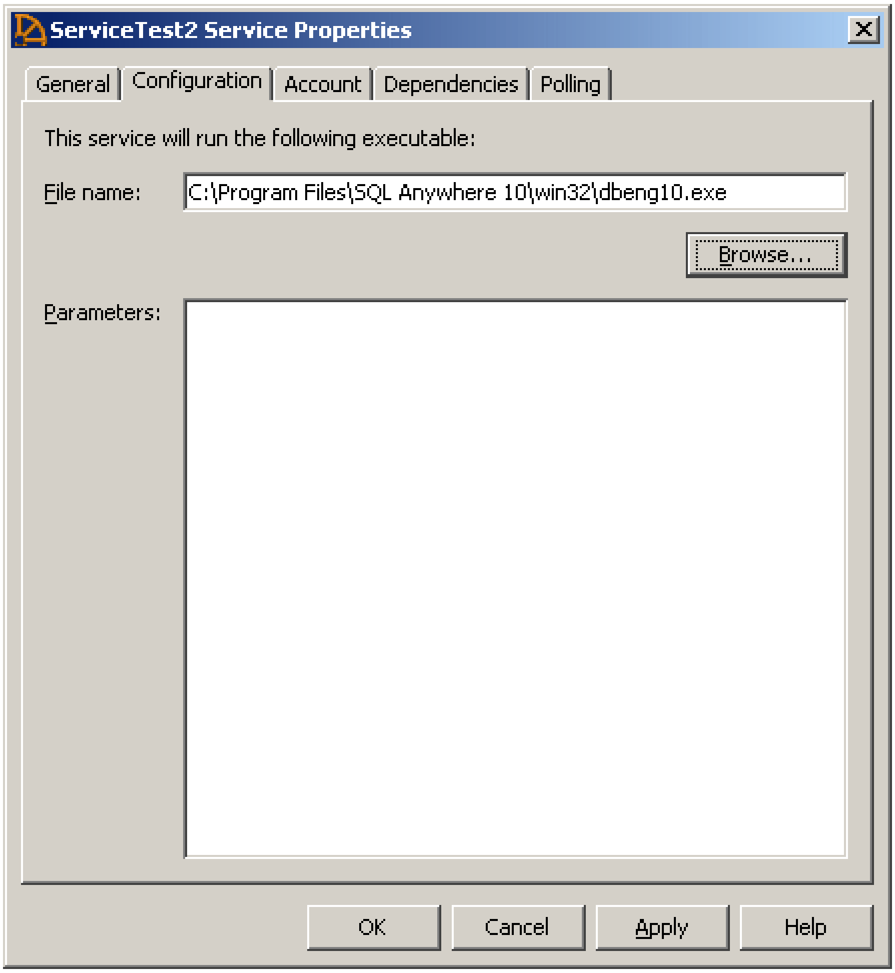
- To start a network server service with a cache size of 20 MB, named my_server running two databases, you would type the following in the Parameters field:

  ```
  -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 property sheet.



This service will run the following executable:

File name: C:\Program Files\SQL Anywhere 10\win32\dbeng10.exe

Parameters:

☞ The options for a service are the same as those for the executable. For a full description of the options for each program, see “The Database Server” on page 117.

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 property sheet, 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 can be granted from the Windows User Manager application under Advanced Privileges. The Service Creation 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 property sheet, an icon appears on the desktop of every user logged in to Windows on the computer running the service. Consequently, 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 property sheet, 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 property sheet 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.

Adding 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

1. From the Context dropdown list, choose the SQL Anywhere 10 plug-in.
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 how to start a database on a server 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*].

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.

Setting the service polling frequency

Sybase Central can poll at specified intervals to check the state (started or stopped) of each service, and update the icons to display the current state. By default, polling is off. If you leave it off, you must click Refresh Folder to see changes to the state.

♦ To set the Sybase Central polling frequency

1. From the Context dropdown list, choose the SQL Anywhere 10 plug-in.
2. In the right pane, click the Services tab.
3. Select the service and then from the File menu, choose Properties.
4. Click the Polling tab.
5. Select Enable Polling.
6. Set the polling frequency.

   The frequency applies to all services, not just the one selected. The value you set in this window remains in effect for subsequent sessions until you change it.
7. Click OK.

Starting and stopping services

♦ To start or stop a service

1. From the Context dropdown list, choose the SQL Anywhere 10 plug-in.
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

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.

Running more than one service

This section describes some topics specific to running more than one service at a time.

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 or Log Transfer Manager to assist in replication.

In cases such as these, the services must start in the proper 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
Replication Agent	SQLANYLTM

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.

♦ To check and change which group a service belongs to

1. From the Context dropdown list, choose the SQL Anywhere 10 plug-in.

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.

Managing 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

1. From the Context dropdown list, choose the SQL Anywhere 10 plug-in.
2. In the right pane, click the Services tab.
3. Select the service, and then choose File ► Properties.

   The Service property sheet appears.
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.

Troubleshooting 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 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. This causes the transaction log file to be different than the database, and results in an invalid transaction log file.

Ensure that you have sufficient 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 sufficient disk space available to the temporary directory, you will have problems starting the server.

☞ For more information, see "SATMP environment variable" on page 285.

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 should be straightforward.

☞ If you experience problems, if you are running non-standard software, or if you are running multiple networks, you may want to read the full discussion of network communication issues in "Client/Server Communications" on page 103.

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 server, you use the -z option. The startup information appears on the Server Messages window: you can use the -o option to log the results to an output file.

☞ For more information, see "-z server option" on page 187 and "-o server option" on page 161.

Make sure you're using the right 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 server information, including server name, protocol, and address. It is possible that the server information in this file is 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. The default location of *sasrv.ini* is *%ALLUSERSPROFILE%\Application Data\SQL Anywhere 10* on Windows and *~/.sqlanywhere10* on Unix.

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 *win32*, *ia64*, or *x64* subdirectory of your SQL Anywhere installation directory (you can find this in the directory listed in the *HKEY_LOCAL_MACHINE\SOFTWARE\Sybase\SQL Anywhere\10.0\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 81.

Create a debug log file

You can use the LogFile connection parameter to create a debug log file. Log files can provide more detailed information about where a connection failure occurred, thereby helping you troubleshoot and correct the problem.

☞ For more information about log files, see "LogFile connection parameter [LOG]" on page 232.

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 (mlsrv)
- network server (dbsrv10)
- personal server (dbeng10)
- QAnywhere agent (qaagent)
- Replication Agent (dbltm)
- 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 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. The location of the error report file is written to the console and the request log.

Error report file names are composed as follows:

- a prefix that identifies the application:

Application identifier	Application
LSN	Listener utility
LTM	Replication Agent
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, *SA10_20051220_133828_32116.mini_core* is an error report from a SQL Anywhere version 10 database server from 2006/06/20, at 1:38:28 pm, from process 32116.

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 SQL Anywhere Support utility (dbsupport) and passes it the name of the error report file to be submitted. By default, dbsupport attempts to prompt the user to submit an error report when it is generated, but if dbsupport is unable to prompt the user, 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.

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 283.

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. See "The SQL Anywhere Support utility" on page 672.

Error reports and diagnostic information are uploaded to the iAnywhere Error Reporting web site via HTTP. This is done to save 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.

CHAPTER 3

Connecting to a Database

Contents

About this chapter

This chapter describes how client applications connect to databases. It contains information about connecting to databases from ODBC, OLE DB, ADO.NET, the iAnywhere JDBC driver, and embedded SQL applications. It also describes connecting from Sybase Central and Interactive SQL.

☞ For more information on connecting to a database from Sybase Open Client applications, see "SQL Anywhere as an Open Server" on page 919.

☞ For more information on connecting via JDBC (if you are not working in Sybase Central or Interactive SQL), see "JDBC API" [*SQL Anywhere Server - Programming*].

Introduction to connections

Any client application that uses a database must establish a **connection** to that database before any work can be done. The connection forms a channel through which all activity from the client application takes place. For example, your user ID determines permissions to perform actions on the database—and the database server has your user ID because it is part of the request to establish a connection.

When a user connects to a database, the database server assigns the 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 -z server output. The connection ID can be used to filter request logging information, identify which connection has a lock on the database, or track the total number of connections to a server since it started and the order in which those connections were made.

☞ For information about request logging, see "Request logging" [*SQL Anywhere Server - SQL Usage*].

☞ For information about locks, see "How locking works" [*SQL Anywhere Server - SQL Usage*].

How connections are established

To establish a connection, the client application calls functions in one of the SQL Anywhere interfaces. SQL Anywhere provides the following interfaces:

- **ODBC** ODBC connections are discussed in this chapter.
- **OLE DB** OLE DB connections are discussed in this chapter.

  For information about OLE DB connections, see "Connecting to a database using OLE DB" on page 71.
- **ADO.NET** ADO.NET connections are not discussed in this chapter.

  For information about ADO.NET connections, see "Connecting to a database" [*SQL Anywhere Server - Programming*].
- **Embedded SQL** Embedded SQL connections are not discussed in this chapter.

  For information about embedded SQL connections, see "SQL Anywhere Embedded SQL" [*SQL Anywhere Server - Programming*].
- **Sybase Open Client** Open Client connections are not discussed in this chapter.

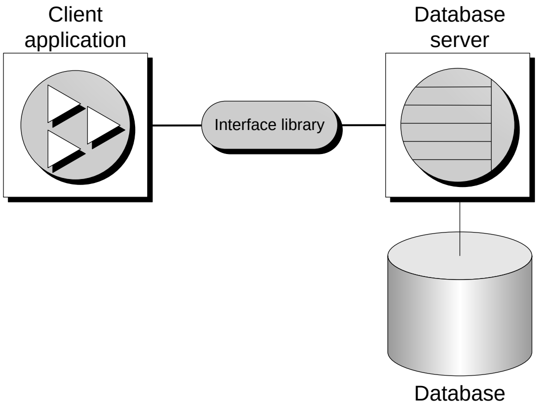
  For information on connecting from Open Client applications, see "SQL Anywhere as an Open Server" on page 919 and "Sybase Open Client API" [*SQL Anywhere Server - Programming*].
- **iAnywhere JDBC driver** JDBC connections using the iAnywhere JDBC driver are not discussed in this chapter.

  For information about iAnywhere JDBC driver connections, see "Establishing JDBC connections" [*SQL Anywhere Server - Programming*].

- **jConnect JDBC driver** JDBC connections using the jConnect JDBC driver are not discussed in this chapter.

  For information on connecting via JDBC, see “Establishing JDBC connections” [*SQL Anywhere Server - Programming*].

The interface uses connection information included in the call from the client application, perhaps together with information held in a data source, the SQLCONNECT environment variable, or the server address cache, to locate and connect to a server running the required database. The following figure is a simplified representation of the pieces involved.



What to read

The following table identifies where you can find answers to questions.

If you want…	Consider reading…
An overview of connecting from Sybase Central or Interactive SQL (including a description of the drivers involved)	“Connecting from Sybase Central, Interactive SQL, or the SQL Anywhere Console utility” on page 54
Some examples to get started quickly, including Sybase Central and Interactive SQL scenarios	“Simple connection examples” on page 56
To learn about data sources	“Working with ODBC data sources” on page 64
To learn what connection parameters are available	“Connection parameters” on page 206
To see an in-depth description of how connections are established	“Troubleshooting connections” on page 75
To learn about network-specific connection issues	“Client/Server Communications” on page 103
To learn about character set issues affecting connections	“Connection strings and character sets” on page 322
To learn about connecting through a firewall	“Connecting across a firewall” on page 106

How connection parameters work

When an application connects to a database, it uses a set of **connection parameters** to define the connection. Connection parameters include information such as the server name, the database name, and a user ID.

A keyword-value pair (of the form *parameter=value*) specifies each connection parameter. For example, you specify the password connection parameter for the default password as follows:

```
Password=sql
```

Connection parameters are assembled into **connection strings**. In a connection string, a semicolon separates each connection parameter, as follows:

```
ServerName=demo10;DatabaseName=demo
```

Representing connection strings

This chapter has many examples of connection strings represented in the following form:

```
parameter1=value1
parameter2=value2
...
```

This is equivalent to the following connection string:

```
parameter1=value1;parameter2=value2
```

You must enter a connection string on a single line with the parameter settings separated by semicolons.

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;...
```

In general, the connection string built by an application and passed to the interface library does not correspond directly to the way a user enters the information. Instead, a user may fill in a dialog, 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 on to the interface library without change. For example, the following is a typical Backup utility (dbbackup) command line:

```
dbbackup -c "ENG=sample_server;DBN=demo;UID=DBA;PWD=sql" SQLAnybackup
```

See also

- ♦ “Connection parameter tips” on page 73

Saving connection parameters in ODBC data sources

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.

☞ For more information on ODBC data sources, see "Working with ODBC data sources" on page 64.

For SQL Anywhere, ODBC data sources can be used by all client interfaces, except Open Client and jConnect. On Unix and Windows CE, the data source is stored in a file. ODBC data sources can not be used on NetWare.

Connecting from Sybase Central, Interactive SQL, or the SQL Anywhere Console utility

To use Sybase Central, Interactive SQL, or the SQL Anywhere Console utility for managing your database, you must first connect to it. In the Connect dialog, you tell Sybase Central, Interactive SQL, or the SQL Anywhere Console what database you want to connect to, where it is located, and how you want to connect to it.

The connection process depends on your situation. For example, if you have a server already running on your computer and this server contains only one database, all you have to do in the Connect dialog is provide a user ID and a password for that database. Sybase Central, Interactive SQL, or the SQL Anywhere Console then knows to connect immediately to the database on the running server.

If this running server has more than one database loaded on it, if it is not yet running, or if it is running on another computer, you need to provide more detailed information in the Connect dialog so that Sybase Central, Interactive SQL, or the SQL Anywhere Console knows which database to connect to.

☞ For more information about connection examples, including examples for Sybase Central and Interactive SQL, see “Simple connection examples” on page 56.

Opening the Connect dialog

A common Connect dialog is available in both Sybase Central and Interactive SQL to let you connect to a database.

When you start Sybase Central, you need to open this dialog manually. When you start Interactive SQL, the dialog automatically appears; you can also make it appear by choosing SQL ► Connect.

♦ To open the Connect dialog (Sybase Central)

- In Sybase Central, choose Connections ► Connect with SQL Anywhere 10.

  You press F11 to open the Connections menu.

Tip
You can make subsequent connections to a given database easier and faster using a **connection profile**.

♦ To open the Connect dialog (Interactive SQL)

- In Interactive SQL, choose SQL ► Connect.

  Alternatively, you can press F11 to open the Connect dialog.

Once the Connect dialog appears, you must specify the connection parameters you need to connect. For example, you can connect to the sample database by choosing SQL Anywhere 10 Demo from the ODBC Data Source Name list on the Identification tab and then clicking OK.

♦ To open the Connect dialog (SQL Anywhere Console utility)

- At a command prompt, type dbconsole.

  The Connect dialog appears.

Working with the Connect dialog

The Connect dialog lets you define parameters for connecting to a server or database. The same dialog is used in both Sybase Central and Interactive SQL. The information entered in the Connect dialog is not preserved between sessions.

The Connect dialog has the following tabs:

- The Identification tab lets you identify yourself to the database and specify a data source.
- The Database tab lets you identify a server and/or database to connect to.
- The Advanced tab lets you add additional connection parameters and specify a driver for the connection.

After you connect successfully in Sybase Central, the database name appears in the left pane of the main window, under the server that it is running on. The user ID for the connection appears after the database name.

In Interactive SQL, the connection information (including the database name, your user ID, and the database server) appears in the title bar above the SQL Statements pane.

Simple connection examples

Although the connection model for SQL Anywhere is configurable, and can become complex, in many cases connecting to a database is very simple.

Who should read this section?

This section describes some simple cases of applications connecting to a SQL Anywhere database. This section may be all you need to get started.

☞ For more information about available connection parameters and their use, see "Connection parameters" on page 206.

Connecting to the sample database from Sybase Central or Interactive SQL

Many examples and exercises throughout the documentation start by connecting to the sample database from Sybase Central or Interactive SQL.

♦ To connect to the sample database (Sybase Central)

1. Start Sybase Central: from the Start menu, choose Programs ► SQL Anywhere 10 ► Sybase Central.
2. Open the Connect dialog: choose Connections ► Connect with SQL Anywhere 10.
3. Select the ODBC Data Source Name option and click Browse.
4. Select SQL Anywhere 10 Demo and then click OK.

♦ To connect to the sample database (Interactive SQL)

1. Start Interactive SQL: from the Start menu, choose Programs ► Sybase ► SQL Anywhere 10 ► Interactive SQL.
2. Open the Connect dialog: from the SQL menu, choose Connect.
3. Select the ODBC Data Source Name option and click Browse.
4. Select SQL Anywhere 10 Demo and then click OK.

   The database name, user ID, and server name appear in the title bar above the SQL Statements pane.

> **Note**
> You do not need to enter a user ID and a password for this connection because the data source already contains this information.

♦ To connect to the sample database (specifying the database file location)

1. In Sybase Central or Interactive SQL, open the Connect dialog.
2. On the Identification tab, type the following values:

- **User ID** Type **DBA**.
- **Password** Type **sql**.

3. On the Database tab, type or browse to the location of the sample database file, *demo.db*. By default, this file is located in *samples-dir*. For example, on Windows XP the default location is *C:\Documents and Settings\All Users\Documents\SQL Anywhere 10\Samples\demo.db*.

   ☞ For more information about the default location of *samples-dir*, see "The samples directory" on page 295.

4. Click OK.

Connecting to a database on your own computer from Sybase Central or Interactive SQL

In the simplest connection scenario, the database you want to connect to resides on your own computer. If this is the case for you, ask yourself the following questions:

- Is the database already running on a server? If not, you need to identify the database file so that Sybase Central or Interactive SQL can start it for you. If so, you can specify fewer parameters in the Connect dialog.
- Are there multiple databases running on your computer? If there is only one, Sybase Central or Interactive SQL assumes that it is the one you want to connect to, and you don't need to specify it in the Connect dialog. If so, you need to tell Sybase Central or Interactive SQL which database in particular to connect to.

The procedures below depend on your answers to these questions.

♦ To connect to a database on an already-running local server

1. Start Sybase Central or Interactive SQL and open the Connect dialog (if it doesn't appear automatically).
2. On the Identification tab of the dialog, enter a user ID and a password for the currently-running database.
3. Do one of the following:
   - If the server is only running one database, click OK to connect to it.
   - If the server is running multiple databases, click the Database tab of the dialog and specify a database name. This is usually the database file name, without the path or extension.

♦ To start and connect to a database

1. Start Sybase Central or Interactive SQL and open the Connect dialog (if it doesn't appear automatically).
2. On the Identification tab of the dialog, enter a user ID and a password.
3. Click the Database tab.

4. Specify a file in the Database File field (including the full path, name, and extension). You can search for a file by clicking Browse.

5. If you want the database name for subsequent connections to be different from the file name, enter a name in the Database Name field (without including a path or extension).

Tips
If the database is already loaded (started) on the server, you only need to provide a database name for a successful connection. The database file is not necessary.
You can connect using a data source (a stored set of connection parameters) for either of the above scenarios by selecting the appropriate data source option at the bottom of the Identification tab of the Connect dialog.

See also

- “Simple connection examples” on page 56

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 largely hidden from the application's user.

When an application uses an embedded database, the personal server is generally not running when the application connects. In this case, you can start the database using the connection string, and by specifying the database file in the DatabaseFile (DBF) parameter of the connection string.

Using the DBF parameter

The DatabaseFile (DBF) parameter specifies which 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.

The following connection parameters show how to load the sample database as an embedded database:

```
DBF=samples-dir\demo.db
UID=DBA
PWD=sql
```

☞ For information about the default location of *samples-dir*, see “The samples directory” on page 295.

Using the ENG parameter

It is recommended that you use the EngineName (ENG) connection parameter when using an embedded database. This ensures that the database will connect to the correct database server in case there are other applications running SQL Anywhere database servers on the same computer.

Using the StartLine [START] parameter

The following connection parameters show how you can 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=dbeng10 -c 8M
DBF=samples-dir\demo.db
UID=DBA
PWD=sql
```

There are a number of 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:

- EngineName (ENG)
- DatabaseFile (DBF)
- DatabaseSwitches (DBS)
- DatabaseName (DBN)

See also

- "Opening the Connect dialog" on page 54
- "Simple connection examples" on page 56

Connecting using a data source

You can save sets of connection parameters in a **data source**. All SQL Anywhere interfaces, except Open Client and jConnect, can use data sources.

♦ To connect using a data source (Sybase Central or Interactive SQL)

1. Start Sybase Central or Interactive SQL and open the Connect dialog (if it doesn't appear automatically).

2. On the Identification tab, enter a user ID and password.

3. On the lower half of the Identification tab, do one of the following:

   - Select the ODBC Data Source Name option and specify a data source name (equivalent to the DataSourceName (DSN) connection parameter, which references a data source in the Windows registry). You can view a list of data sources by clicking Browse.

   - Select the ODBC Data Source File option and specify a data source file (equivalent to the FileDataSourceName (FILEDSN) connection parameter, which references a data source held in a file). You can search for a file by clicking Browse.

The SQL Anywhere 10 Demo data source holds a set of connection parameters, including the database file and a StartLine (START) parameter to start the database.

See also

- "Opening the Connect dialog" on page 54

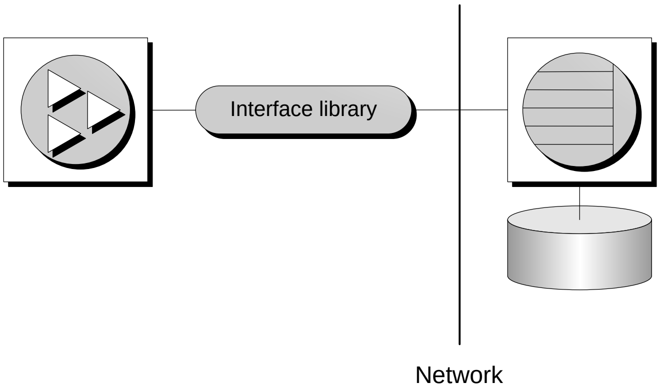
- ♦ “Simple connection examples” on page 56
- ♦ “Using ODBC data sources on Unix” on page 67

Connecting to a server on a network

To connect to a database running on a network server somewhere on a local or wide area network, the client software must locate and connect to the database server. SQL Anywhere provides a network library to handle this task.

Network connections occur over a **network protocol**. TCP/IP is available on all platforms, and SPX is available on some platforms.

☞ For more information about client/server communications over a network, see “Client/Server Communications” on page 103.



Specifying the server

SQL Anywhere server names must be unique on a local domain for a given network protocol. The following connection parameters provide a simple example for connecting to a server running elsewhere on a network:

```
ENG=svr_name
DBN=db_name
UID=user_id
PWD=password
CommLinks=all
```

When CommLinks=all is specified, the client library first looks for a personal server of the given name, and then looks on the network for a server of the given name.

☞ For information, see “CommLinks connection parameter [LINKS]” on page 211.

Specifying the protocol

If several protocols are available, you can instruct the network library which ones to use to improve performance. The following parameters use only the TCP/IP protocol:

```
ENG=svr_name
DBN=db_name
UID=user_id
PWD=password
CommLinks=tcpip
```

The network library searches for a server by broadcasting over the network, which can be a time-consuming process. Once the network library locates a server, the client library stores its name and network address in a file (*sasrv.ini*), and reuses this entry for subsequent connection attempts to that server using the specified protocol. Subsequent connections are normally faster than a connection achieved by broadcast.

Many other connection parameters are available to assist SQL Anywhere in locating a server efficiently over a network.

☞ For more information, see "Network protocol options" on page 242.

☞ For more information, see "Connecting across a firewall" on page 106.

♦ **To connect to a database on a network server (Sybase Central or Interactive SQL)**

1. Start Sybase Central or Interactive SQL and open the Connect dialog (if it doesn't appear automatically).
2. On the Identification tab of the dialog, enter a user ID and a password.
3. On the Database tab of the dialog, enter the Server Name. You can search for a server by selecting the Search Network for Database Servers option and then clicking Find.
4. Identify the database by specifying a Database Name.

Tips
You can connect using a data source (a stored set of connection parameters) by selecting the appropriate data source option at the bottom of the Identification tab of the Connect dialog.
By default, all network connections in Sybase Central and Interactive SQL use the TCP/IP network protocol.

See also

♦ "Opening the Connect dialog" on page 54
♦ "Simple connection examples" on page 56

Using default connection parameters

You can leave many connection parameters unspecified, and instead use the default behavior to make a connection. Be cautious about relying on default behavior in production environments, especially if you distribute your application to customers who may install other SQL Anywhere applications on their computer.

Default database server and database

If a single personal server is running, with a single loaded database, you can connect using entirely default parameters:

```
UID=user-id
PWD=password
```

Default database server

If more than one database is loaded on a single personal server, you can leave the server as a default, but you need to 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, you need to specify which server you want to connect to. If only one database is loaded on that server, you do not need to specify the database name. The following connection string connects to a named server, using the default database:

```
ENG=server-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:

```
ENG=server-name
DBN=db-name
UID=user-id
PWD=password
```

☞ For more information about default behavior, see "Troubleshooting connections" on page 75.

No defaults for a network server

To connect to a network server running on a different computer:

```
ENG=server-name
DBN=dbn
UID=user-id
PWD=password
CommLinks=tcpip
```

If CommLinks 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 select the Search Network for Database Servers option on the Connect dialog to attempt a network connection.

☞ For more information about default behavior, see "Troubleshooting connections" on page 75.

Connecting from SQL Anywhere utilities

All SQL Anywhere database utilities that communicate with the server (rather than acting directly on database files) do so using embedded SQL. They follow the procedure outlined in "Troubleshooting connections" on page 75 when connecting to a database.

How database tools obtain connection parameter values

Many of the administration utilities obtain the connection parameter values by:

1. Using values specified on the command line (if there are any). 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 options for each database tool, see the chapter “Database Administration Utilities” on page 585.

2. Using the SQLCONNECT environment variable settings if any values are missing. SQL Anywhere does not set this variable automatically.

   ☞ For more information about the SQLCONNECT environment variable, see “SQLCONNECT environment variable” on page 288.

Working with ODBC data sources

Microsoft defines the **Open Database Connectivity** (**ODBC**) interface, which is a standard interface for connecting client applications to database management systems in the Windows XP/200x environments. Many client applications, including application development systems, use the ODBC interface to access a wide range of database systems.

The SQL Anywhere ODBC driver is named *dbodbc10.dll*, and it is located in *install-dir\win32*.

☞ 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 any of the following applications:

- Sybase Central and Interactive SQL.
- All the SQL Anywhere utilities.
- 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.

Where data sources are held

You connect to an ODBC database using an ODBC data source. You need an ODBC data source on the client computer for each database you want to connect to.

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.

If you have a data source, your connection string can simply name the data source to use:

- **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
  ```

For SQL Anywhere, the use of ODBC data sources goes beyond Windows applications using the ODBC interface:

- SQL Anywhere client applications on Unix can use ODBC data sources, as well as those on Windows operating systems.
- ODBC data sources can be used by all SQL Anywhere client interfaces except jConnect and Open Client.

Note

When creating a connection string, it can contain the name of an ODBC data source that contains connection parameters, as well as connection parameters that are specified explicitly. If a connection parameter is specified in the connection string as well as in the ODBC data source, the value that is specified explicitly takes precedence.

Creating an ODBC data source

You can create ODBC data sources on Windows XP/200x operating systems using the ODBC Administrator, which provides a central place for creating and managing ODBC data sources.

SQL Anywhere also includes a cross-platform utility named dbdsn to create data sources.

Creating ODBC data sources using the ODBC Administrator

On Windows XP/200x, you can use the Microsoft ODBC Administrator to create and edit data sources. You can work with User Data Sources, File Data Sources, and System Data Sources in this utility.

♦ To create an ODBC data source (ODBC Administrator)

1. Start the ODBC Administrator by choosing Start ► Programs ► SQL Anywhere 10 ► SQL Anywhere ► ODBC Administrator.

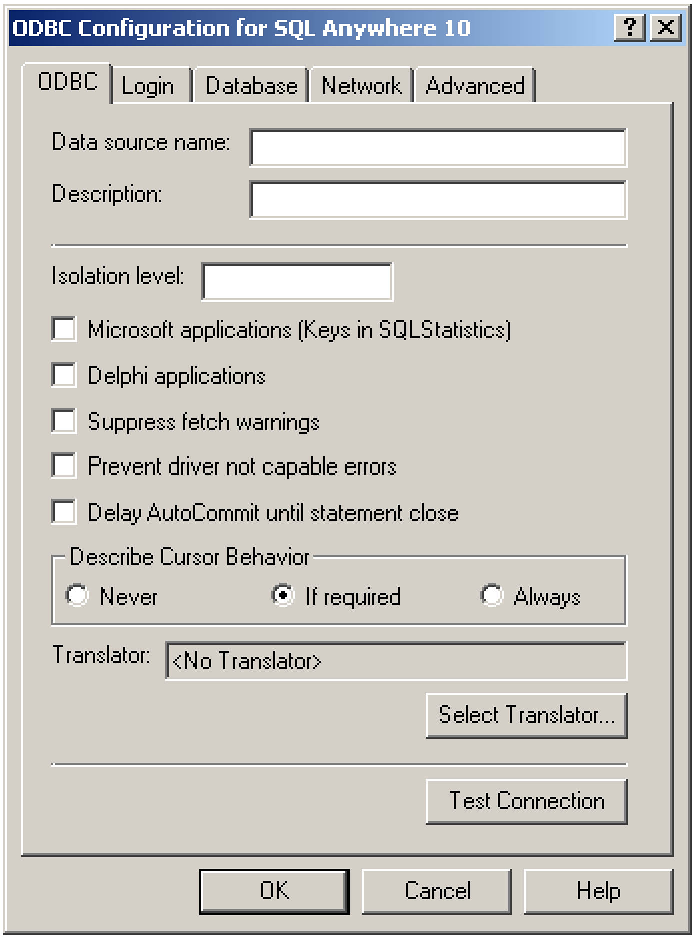
   The ODBC Data Source Administrator dialog appears.

2. Click Add.

   The Create New Data Source wizard appears.

3. From the list of drivers, choose SQL Anywhere 10, and then click Finish.

   The ODBC Configuration for SQL Anywhere dialog appears.



Data source name:

Description:

Isolation level:

□ Microsoft applications (Keys in SQLStatistics)

□ Delphi applications

□ Suppress fetch warnings

□ Prevent driver not capable errors

□ Delay AutoCommit until statement close

Describe Cursor Behavior

○ Never ● If required ○ Always

Translator: <No Translator>

☞ For descriptions of the fields in the dialog, see “ODBC Configuration Dialog Help” [*SQL Anywhere 10 - Context-Sensitive Help*].

4. When you have specified the parameters you need, click OK to close the dialog and create the data source.

Editing an ODBC data source using the ODBC Administrator

To edit a data source, find and select the desired data source in the ODBC Administrator main window and then click Configure.

Creating an ODBC data source from the command line

You can create User and System Data Sources using the dbdsn utility. You can not create File Data Sources in this way. System Data Sources are limited to Windows operating systems only, and you can use the ODBC Administrator to create File Data Sources.

♦ To create an ODBC data source (Command line)

1. Open a command prompt.

2. Enter 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" "UID=DBA;PWD=sql;DBF=samples-dir\demo.db"
   ```

   ☞ For information about the default location of *samples-dir*, see "The samples directory" on page 295.

☞ For more information on the dbdsn utility, see "The Data Source utility" on page 595.

Using file data sources on Windows

On Windows operating systems, ODBC data sources are typically 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.

File data sources can be distributed

One benefit of file data sources is that you can distribute the file to users. If the file is placed in the default location for file data sources, it is picked up automatically by ODBC. In this way, managing connections for many users can be made simpler.

♦ To create an ODBC file data source (ODBC Administrator)

1. Start the ODBC Administrator, click the File DSN tab and then click Add.

2. Select SQL Anywhere 10 from the list of drivers, and then click Next.

3. Follow the instructions to create the data source.

Using ODBC data sources on Unix

On Unix operating systems, ODBC data sources are held in a system information file. By default, this file is named *.odbc.ini*, but it can have any name. The following locations are searched, in order, for the system information file:

1. The ODBCINI environment variable.

2. The ODBC_INI environment variable.

3. The ODBCHOME environment variable.

4. The HOME environment variable.
5. The user's Home directory (~).
6. The PATH environment variable.

Note
The ODBCINI and ODBC_INI environment variables point to the system information file (which may or may not be named *.odbc.ini*), while the ODBCHOME and HOME environment variables point to a 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]
ENG=myserver
CommLinks=tcpip(Host=hostname)
UID=DBA
PWD=sql
```

You can enter any connection parameter in the system information file. See "Connection parameters" on page 206.

Network protocol options are added as part of the CommLinks (LINKS) parameter. See "Network protocol options" on page 242.

You can create and manage ODBC data sources on Unix using the dbdsn utility.

☞ For more information, see "Creating an ODBC data source" on page 65, and "The Data Source utility" on page 595.

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 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.

Using ODBC data sources on Windows CE

Windows CE does not provide an ODBC driver manager or an ODBC Administrator. On this platform, SQL Anywhere uses ODBC data sources stored in files. You can specify either the DSN or the FILEDSN keyword to use these data source definitions—on Windows CE (only), DSN and FILEDSN are synonyms.

Data source location

Windows CE searches for the data source files in the following locations:

1. The directory from which the ODBC driver (*dbodbc10.dll*) was loaded. This is usually the Windows directory.
2. The directory specified in Location key of the SQL Anywhere section of the Windows CE registry. This is usually the root directory.

```
\filename.dsn
```

Each data source itself is held in a file. The file has the same name as the data source, with an extension of *.dsn*.

☞ For more information about file data sources, see "Using file data sources on Windows" on page 67.

A sample Windows CE data source

The following is a sample of an ODBC data source for Windows CE.

```
[ODBC]
DRIVER=\windows\dbodbc10.dll
UID=DBA
PWD=sql
Integrated=No
AutoStop=Yes
EngineName=SalesDB_remote
LINKS=tcpip(host=192.168.0.55;port=2638;dobroadcast=none)
LOG=\sa_connection.txt
START=dbsrv10 -c 8M
```

See also

- "Creating an ODBC data source to connect to your Windows CE device" on page 973

Connecting from desktop applications to a Windows CE database

You can connect from applications running on a desktop PC, such as Sybase Central or Interactive SQL, to a database server running on a Windows CE device. The connection uses TCP/IP over the ActiveSync link between the desktop computer and the Windows CE device.

☞ For information about connecting to a Windows CE database, see:

- “Determining the IP address of your Windows CE device” on page 971
- “Creating proxy ports for Windows CE devices” on page 972
- “Creating an ODBC data source to connect to your Windows CE device” on page 973
- “Connecting to a database running on a Windows CE device” on page 971

Connecting to a database using OLE DB

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.

This section describes how to connect to a SQL Anywhere database using OLE DB from 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 PowerBuilder OLE DB database profile.

This section is an introduction to how to use OLE DB from Sybase PowerBuilder and Microsoft ADO environments such as Visual Basic. It is not complete documentation on how to program using ADO or OLE DB. The primary source of information on development topics is your development tool documentation.

☞ For more information about OLE DB, see "Introduction to OLE DB" [*SQL Anywhere Server - Programming*].

OLE DB providers

You need an OLE DB provider for each type of data source you want to access. Each **OLE DB provider** is a dynamic-link library. There are two OLE DB providers you can use to access SQL Anywhere:

- **Sybase 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**.

  When the **SAOLEDB** provider is installed, it registers itself. 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.

  For more information about OLE DB providers, see "Introduction to OLE DB" [*SQL Anywhere Server - Programming*].

- **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.

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.

☞ For more information about ADO, see "ADO programming with SQL Anywhere" [*SQL Anywhere Server - Programming*].

Example

The following 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 10 Demo"
myConn.Open
```

Connection parameter tips

Connection parameters often provide more than one way of accomplishing a given task. This is particularly the case with embedded databases, where the connection string starts a database server. For example, if your connection string starts a database, you can specify the database name using the DatabaseName (DBN) connection parameter or using the DatabaseSwitches (DBS) parameter. The Database Name (DBN) connection parameter is recommended.

Here are some recommendations and notes for situations where connection parameters conflict:

- **Specify database files using DBF** You can specify a database file in the StartLine (START) parameter or using the DatabaseFile (DBF) connection parameter (recommended).
- **Specify database names using DBN** You can specify a database name in the StartLine (START) parameter, the DatabaseSwitches (DBS) connection parameter, or using the DatabaseName (DBN) connection parameter (recommended).
- **Use the Start parameter to specify cache size** Even though you use the DatabaseFile (DBF) connection parameter to specify a database file, you may still want to tune the way in which it starts. You can use the StartLine (START) connection parameter to do this.

  For example, suppose you want to provide additional cache memory in the StartLine (START) connection parameter. The following set of embedded database connection parameters starts a database server with extra cache:

  ```
  DBF=samples-dir\demo.db
  DBN=Sample
  ENG=Sample Server
  UID=DBA
  PWD=sql
  START=dbeng10 -c 8M
  ```

☞ For a list of connection parameters, see "Connection Parameters and Network Protocol Options" on page 205.

☞ For character set issues in connection strings, see "Connection strings and character sets" on page 322.

Notes about connection parameters

- **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).
- The connection parameters used by the interface library can be obtained from the following places (in order of precedence):
  - **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 server name must be composed of the ASCII character set in the range 1 to 127. There is no such limitation on other parameters.
  For more information on the character set issues, see "Connection strings and character sets" on page 322.

- **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 in case of conflict the explicit parameters are used.

- **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.

Troubleshooting connections

Who needs to read this section?

In many cases, establishing a connection to a database is straightforward using the information presented in the first part of this chapter.

However, if you are having problems establishing connections to a server, you may need to understand the process by which SQL Anywhere establishes connections to resolve your problems. This section describes how SQL Anywhere connections work.

☞ For more information about network-specific issues, including connections across firewalls, see "Client/ Server Communications" on page 103.

The software follows exactly the same procedure for each of the following types of client application:

- **ODBC** Any ODBC application using the SQLDriverConnect function, which is the common method of connection 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.
- **Embedded SQL** 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 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.
- **OLE DB** 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**.
- **JDBC** Applications using the iAnywhere JDBC driver pass the URL **jdbc:odbc:** followed by a standard connection string as a parameter to the Driver Manager.GetConnection method. The connection string must include **DataSource=** and name a SQL Anywhere data source or include **Driver=SQL Anywhere 10** (this parameter is specified as **Driver=libdbodbc10** on Unix and Linux).

☞ For more troubleshooting tips, see "Troubleshooting server startup" on page 44, and "Troubleshooting network communications" on page 114.

The steps in establishing a connection

To establish a connection, SQL Anywhere performs the following steps in order:

1. **Locate the interface library** The client application must locate the interface library.

2. **Assemble a list of connection parameters** Since connection parameters may appear in several places (such as data sources, a connection string assembled by the application, or an environment variable) SQL Anywhere assembles the parameters into a single list.
3. **Locate a server** Using the connection parameters, SQL Anywhere locates a database server on your computer or over a network.
4. **Locate the database** SQL Anywhere locates the database you want to connect to.
5. **Start a personal server** If SQL Anywhere fails to locate a server, it attempts to start a personal database server and load the database.

The following sections describe each of these steps in detail.

Locating the interface library

The client application makes a call to one of the SQL Anywhere interface libraries. In general, the location of this DLL or shared library is transparent to the user. This section describes how to locate the library in case of problems.

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 looks in the supplied data source to locate the driver. When you create a data source using the ODBC Administrator or dbdsn utility, SQL Anywhere fills in the current location for your ODBC driver. The data source information is stored in the registry on Windows, or in a file on Unix (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 as follows:

- ♦ **Windows XP/200x** *dblib10.dll*
- ♦ **Unix** *dblib10* with an operating-system-specific extension
- ♦ **NetWare** *dblib10.nlm*

OLE DB driver location

The provider name (SAOLEDB) is used to locate the SQL Anywhere OLE DB provider DLL (*dboledb10.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.

iAnywhere JDBC driver location

The Java package *jodbc.jar* must be in the classpath when you run your application. The native DLLs or shared objects must be found by the system.

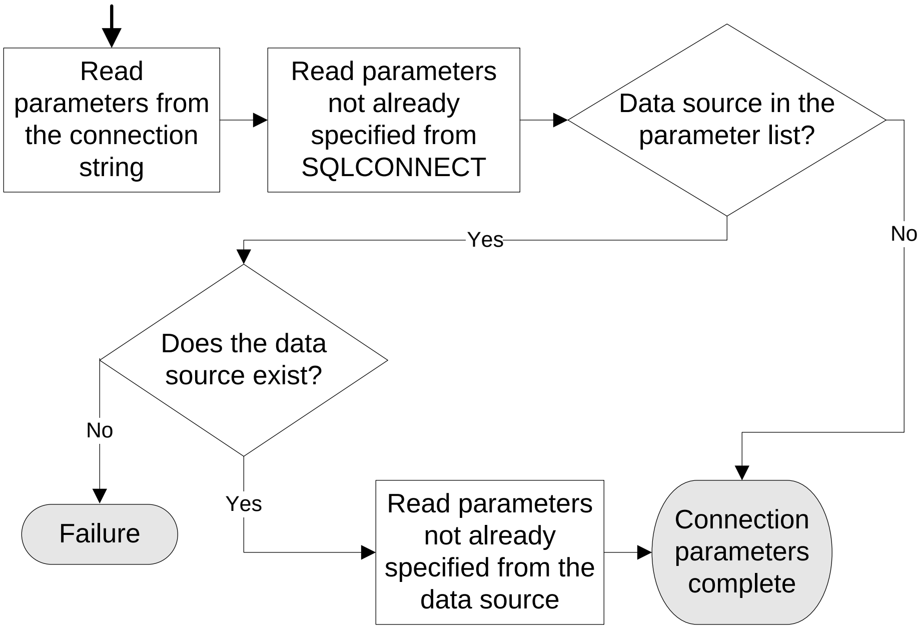
- **PC operating systems** On PC operating systems such as Windows, files are looked for in the current directory, in the system path, and in the *Windows* and *Windows\system* directories.
- **Unix operating systems** On Unix, files are looked for in the system path and the user library path.
- **NetWare** On NetWare, files are looked for in the search path, and in the *sys:system* directory.

When the library is located

Once the client application locates the interface library, it passes a connection string to it. The interface library uses the connection string to assemble a list of connection parameters, which it uses to establish a connection to a server.

Assembling a list of connection parameters

The following figure illustrates how the interface libraries assemble the list of connection parameters they use to establish a connection.


Read parameters from the connection string
Read parameters not already specified from SQLCONNECT
Data source in the parameter list?
Yes
No
Does the data source exist?
No
Yes
Failure
Read parameters not already specified from the data source
Connection parameters complete

Notes

Key points from the figure include:

- **Precedence** Parameters held in more than one place are subject to the following order of precedence:

1. Connection string
2. SQLCONNECT
3. Data source

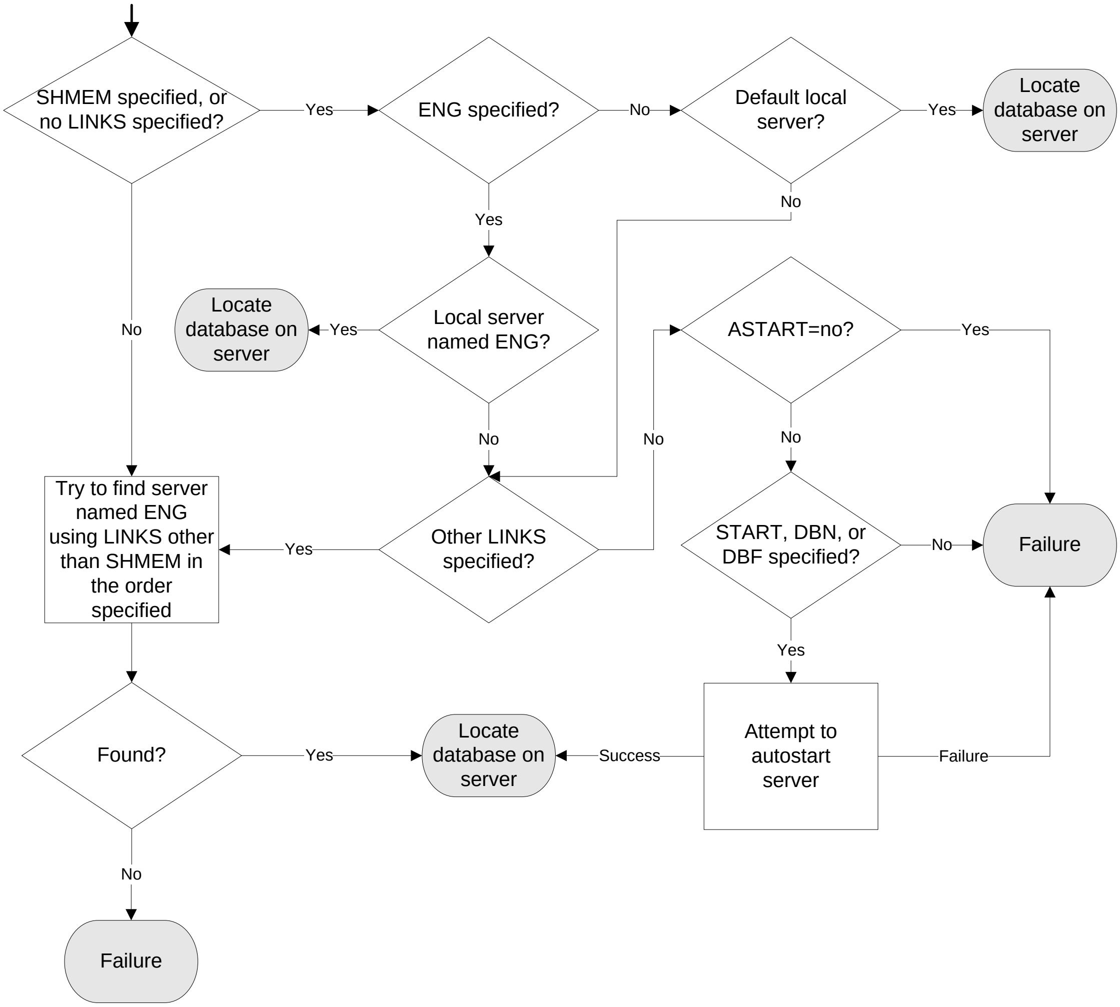
   That is, if a parameter is supplied both in a data source and in a connection string, the connection string value overrides the data source value.

- **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 server

In the next step towards establishing a connection, SQL Anywhere attempts to locate a server. If the connection parameter list includes a server name (EngineName (ENG) connection parameter), it performs a search for a server of that name. If no EngineName (ENG) connection parameter is supplied, and LINKS is not specified or LINKS includes Shared Memory, SQL Anywhere looks for a default server.


SHMEM specified, or no LINKS specified?
Yes
ENG specified?
No
Default local server?
Yes
Locate database on server
No
Yes
No
Local server named ENG?
Yes
Locate database on server
ASTART=no?
Yes
No
No
No
Try to find server named ENG using LINKS other than SHMEM in the order specified
Yes
Other LINKS specified?
START, DBN, or DBF specified?
No
Failure
Yes
Found?
Yes
Locate database on server
Success
Attempt to autostart server
Failure
No
Failure

If SQL Anywhere locates a server, it tries to locate or load the required database on that server.

☞ For information, see "Locating the database" on page 80.

If SQL Anywhere can not locate a server, it may attempt to start a personal server, depending on the connection parameters.

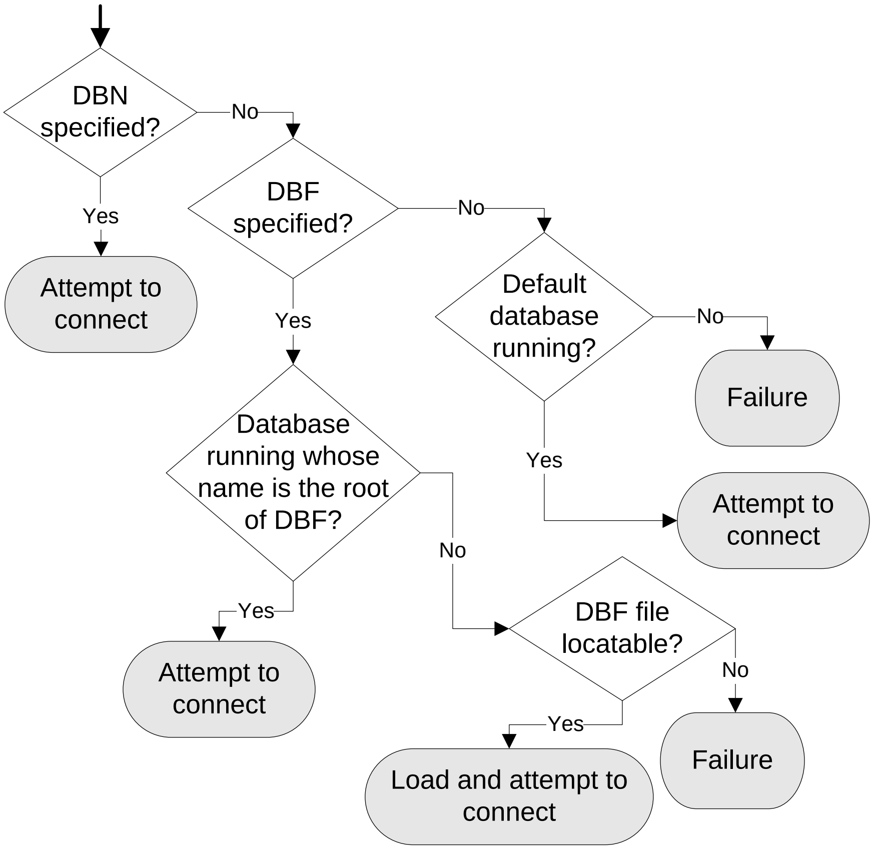
Notes

- 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.

- 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 autostarted, information from the START, DBF, DBKEY, DBS, DBN, ENG, and AUTOSTOP connection parameters are used to construct the options for the autostarted 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 database option" on page 200.

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.

☞ For more information about network protocol options, see "Network protocol options" on page 242.

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 10* on Windows and *~/.sqlanywhere10* 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

The dbping utility is provided to help in troubleshooting connections. In particular, you can use it to test if a server with a particular name is available on your network.

The dbping utility takes a connection string as an option, but by default only those pieces required to locate a server are used. By default, it does not attempt to start a server, but may if the -d option is specified.

Examples

The following command line tests to see if a server named Waterloo is available over a TCP/IP connection:

```
dbping -c "ENG=Waterloo;CommLinks=tcpip"
```

The following command tests to see if a default server is available on the current computer.

```
dbping
```

☞ For more information on dbping options, see "The Ping utility" on page 643.

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;ENG=sampleserver; LINKS=TCPIP"`:

```
SQL Anywhere Server Ping Utility Version 10.0.0.1658
Connected to SQL Anywhere 10.0.0.1657 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
Receive throughput              10100 KB          1024 msec      9863 KB/sec
Ping database successful.
```

See also

- "Ping utility options" on page 644

Using integrated logins

The **integrated login** feature allows you to maintain a single user ID and password for both database connections and operating system and/or network logins. This section describes the integrated login feature.

Supported operating systems

Integrated login capabilities are available for database servers running on Windows XP/200x. It is possible for Windows XP/200x clients to use integrated logins to connect to a network server running on Windows XP/200x.

Benefits of integrated logins

An integrated login is a mapping from one or more Windows users or Windows user group profiles to an existing user in a database. 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 accomplish 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 and/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. Its advantages include:

- The user does not need to type a user ID or password.
- User authentication is done by the operating system, not the database: 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 XP/200x 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 91 and “Security concerns: Copied database files” on page 102.

Creating integrated logins for Windows user groups

In addition to creating integrated logins for individual Windows users, you can create 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.
For ways to prevent particular users that are members of a Windows user group from connecting to a database, see “Preventing members of Windows user groups from connecting to a database” on page 86.

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 of 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.

`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 the SQL Anywhere database server is running on is used to verify Windows user group membership. If the Domain Controller server is a different computer than the one the database server is running on, you can specify the name of the Domain Controller server using the integrated_server_name option. For example:

```
SET PUBLIC.OPTION integrated_server_name = '\\myserver-1'
```

☞ For more information, see “integrated_server_name option [database]” on page 418.

Preventing members of Windows user groups from connecting to a database

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 database user ID or password. There are two methods you can use to prevent a user who is a member of a Windows user group that has 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.

You can use either of these methods to prevent members of Windows user groups from connecting to a database.

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

1. Add a user to the database without a password. For example:

   ```
   GRANT CONNECT TO 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 a stored procedure to be called 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
```

Setting up integrated logins

Several steps must be implemented in order to connect successfully via an integrated login.

♦ To use an integrated login

1. Enable the integrated login feature in a database by setting the value of the login_mode database option to include Integrated (the option is case insensitive), in place of the default value of Standard. This step requires DBA authority.

2. Create a database user to map the integrated login to (if one does not already exist). This can be done using a SQL statement or a wizard in Sybase Central. This step requires DBA authority.

3. Create an integrated login mapping between a Windows user or group profile and an existing database user. This can be done using a SQL statement or a wizard in Sybase Central. In Sybase Central, all users with integrated login permission appear in the Login Mappings folder. This step requires DBA authority.

4. Connect from a client application in such a way that the integrated login facility is triggered.

Each of these steps is described in the sections below.

Enabling the integrated login feature

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 “Using Kerberos authentication” on page 93.

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.

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 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.

Only users with DBA authority are able to create or delete an integrated login mapping.

An integrated login mapping is made either using a wizard in Sybase Central or a SQL statement.

♦ To map an integrated login (Sybase Central)

1. Connect to a database as a user with DBA authority.
2. Click the Login Mappings folder in the left pane.
3. From the File menu, choose New ▶ Login Mapping.
4. On the second page of the wizard, specify the name of the system (computer) user or group profile for whom the integrated login is to be created.

   Also, select the database user ID this user maps to. The wizard displays the available database users. You must select one of these. You can not add a new database user ID.
5. Follow the remaining instructions in the wizard.

♦ To map an integrated login (SQL)

1. Connect to a 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
```

☞ For more information, 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
```

☞ For more information about creating integrated logins for Windows groups, see “Creating integrated logins for Windows user groups” on page 84.

Revoking integrated login permission

You can remove an integrated login mapping using either Sybase Central or Interactive SQL.

♦ To revoke an integrated login permission (Sybase Central)

1. Connect to a database with DBA authority.
2. Open the Login Mappings folder.
3. In the right pane, select the login mapping you want to remove, and then choose File ► Delete.

♦ To revoke an integrated login permission (SQL)

1. Connect to a database 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
```

☞ For more information, see the “REVOKE statement” [*SQL Anywhere Server - SQL Reference*].

Connecting from a client application

A client application can connect to a database using an integrated login in one of the following ways:

- Set the Integrated (INT) parameter in the list of connection parameters to YES.
- Specify neither a user ID nor a password in the connection string or Connect dialog.

If Integrated=YES is specified in the connection string, an integrated login is attempted. If the connection attempt fails and the login_mode database option is set to Standard,Integrated, then the server attempts a standard login. See "login_mode option [database]" on page 422.

If an attempt to connect to a database is made without providing a user ID or password, an integrated login is attempted. The attempt succeeds or fails depending on whether the current user profile name matches an integrated login mapping in the database.

Interactive SQL examples

For example, a connection attempt using the following Interactive SQL statement will succeed, providing the user has logged on with a user profile name that matches an integrated login mapping in a default database of a server:

```
CONNECT USING 'INTEGRATED=yes'
```

The Interactive SQL statement CONNECT can connect to a database if all of the following are true:

- 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.
- If the user is prompted with a dialog by the server for more connection information (such as occurs when using Interactive SQL), the user clicks OK *without* providing more information.

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 exist on both the local computer and the server. As well as having identical user profile names on both computers, 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

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 XP/200x 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 XP/200x 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 XP/200x enforces login security in order 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 XP/200x computer hosting a SQL Anywhere database. This can be done using the Windows User Manager utility.

Using Kerberos authentication

The Kerberos login feature allows you to maintain a single user ID and password for both database connections and operating system and/or network logins. This section describes the Kerberos login feature.

Benefits of Kerberos logins

Kerberos is a network authentication protocol that provides strong authentication and encryption using secret-key cryptography. SQL Anywhere can use Kerberos for authentication in a manner similar to Windows integrated logins. Users who have already logged in to Kerberos can connect to a database without providing a user ID or password.

To use Kerberos as an authentication system, you must configure SQL Anywhere to delegate authentication to Kerberos. The database must be configured to use Kerberos logins, and a mapping must have been granted between the user used to log in to the computer and/or network, and a database user.

If you already have Kerberos set up, then you can take advantage of this authentication mechanism to authenticate users connecting to databases.

Using a Kerberos login is more convenient for the user 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.

Caution
Kerberos 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: Copied database files" on page 102.

SQL Anywhere does not come equipped with Kerberos software. You must obtain Kerberos software separately. Kerberos software includes the following components:

- **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.
- **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, as well as 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 XP/200x use 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 "Using SSPI for Kerberos logins on Windows" on page 98.

Kerberos clients

Kerberos authentication is available on 32-bit Windows XP/200x and Linux. For a list of tested Kerberos clients, see SQL Anywhere Supported Kerberos Clients.

The following table lists the default names and locations of the keytab and GSS-API files used by the supported Kerberos clients.

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.
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.

Setting up Kerberos authentication

The following procedure describes how to configure and use Kerberos authentication with SQL Anywhere.

♦ To set up Kerberos authentication for a SQL Anywhere database

1. Install and configure the Kerberos client software, including the GSS-API runtime library, on both the client and server computers. (If you presently use Kerberos, this should already be done).

   On Windows client computers using an Active Directory KDC, SSPI can be used and no Kerberos client needs to be installed. See "Using SSPI for Kerberos logins on Windows" on page 98.

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.

   Servers, like users, must authenticate themselves to the KDC. This is why you must create a server service principal within the KDC. Servers authenticate themselves through the use of a keytab file. The keytab file is a protected, encrypted file that the server uses to identify itself to the KDC.

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.

Once you have installed and configured Kerberos, you must enable Kerberos logins for your SQL Anywhere database. 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.

The login_mode database option accepts one or more of 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.
- **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. The only exception to this is users with DBA authority (full administrative rights).

The following steps assume you already performed the steps in the previous procedure to set up Kerberos.

♦ To configure SQL Anywhere to use Kerberos

1. Start the SQL Anywhere server with the -krb or -kr option to enable Kerberos authentication (alternatively, the -kl option can be used to specify the location of the GSS-API library and enable Kerberos). See “-kl server option” on page 156, “-kr server option” on page 157, and “-krb server option” on page 157.

   The following command starts the database server for the Kerberos principal my_server_princ@MYREALM.COM.

   ```
   dbsrv10 -krb -n my_server_princ C:\kerberos.db
   ```

2. 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. See “login_mode option [database]” on page 422.

   ```
   SET OPTION PUBLIC.login_mode = 'Kerberos,Standard'
   ```

3. 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.

   ```
   GRANT CONNECT TO "kerberos-user"
   IDENTIFIED BY "abc123"
   ```

4. 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). See “GRANT statement” [*SQL Anywhere Server - SQL Reference*] and “Creating Kerberos login mappings” on page 97.

   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 90.

5. 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 sufficient 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.

6. Connect from the client, specifying the KERBEROS connection parameter (KERBEROS=YES in many cases, 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;ENG=my_server_princ"
   ```

Interactive SQL example

For example, a connection attempt using the following Interactive SQL statement will 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=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 dialog by the server for more connection information (such as occurs when using the Interactive SQL utility), the user clicks OK *without* providing more information.

Open Client and jConnect connections

To connect from an Open Client or jConnect application, follow the previous steps (in the procedure To configure SQL Anywhere to use Kerberos), and set up Open Client/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. Note you cannot connect using an alternate server name from 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 server option” on page 157
- “-kr server option” on page 157
- “-kl server option” on page 156
- “login_mode option [database]” on page 422
- “GRANT statement” [*SQL Anywhere Server - SQL Reference*]
- “Kerberos connection parameter [KRB]” on page 229
- “Troubleshooting Kerberos connections” on page 99

Creating Kerberos login mappings

You can create a Kerberos login mapping using either Sybase Central or a SQL statement.

♦ To create a Kerberos login mapping (Sybase Central)

1. Connect to a database with DBA authority.
2. Open the Login Mappings folder.
3. From the File menu, choose New ► Login Mapping.

   The Create Login Mapping wizard appears.

♦ To create a Kerberos login mapping (SQL)

1. Connect to a database 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"
```

Revoking Kerberos login permission

You can remove a Kerberos login mapping using either Sybase Central or a SQL statement.

♦ To revoke a Kerberos login mapping (Sybase Central)

1. Connect to a database as a user with DBA authority.
2. Open the Login Mappings folder.
3. In the right pane, select the login mapping you want to remove, and then choose File ► Delete.

♦ To revoke a Kerberos login mapping (SQL)

1. Connect to a database 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"
```

Using SSPI for Kerberos logins on Windows

On Windows client computers using a Windows domain, SSPI can be used and no Kerberos client needs to be installed on the client computer. Windows domain accounts already have Kerberos principals associated with them. For example, account pchin in the myrealm.com Windows domain is typically already associated with the pchin@MYREALM.COM Kerberos principal. Clients authenticating this way must set the Kerberos (KRB) connection parameter to SSPI.

When Kerberos=SSPI is specified in the connection string, a Kerberos login is attempted.

The following procedure assumes you have already set up Kerberos authentication using the procedure in "Setting up Kerberos authentication" on page 94.

♦ To connect using SSPI

1. Start the SQL Anywhere server with the -krb option to enable Kerberos authentication.

   ```
   dbeng10 -krb -n my_server_princ C:\kerberos.db
   ```

2. 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.

   ```
   SET OPTION PUBLIC.login_mode = 'Kerberos'
   ```

3. 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.

   ```
   GRANT CONNECT TO "kerberos-user"
   IDENTIFIED BY "abc123"
   ```

4. 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).

   ```
   GRANT KERBEROS LOGIN TO "pchin@MYREALM.COM"
   AS USER "kerberos-user"
   ```

5. Connect to the database from the client computer.

   ```
   dbisql -c "KERBEROS=SSPI;ENG=my_server_princ"
   ```

   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 both 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 will include additional diagnostic messages in the server console. 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.

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 machine, including the GSS-API library ♦ The database server -z output lists the name of the library that it is attempting to load. Verify that the library name is correct, and use the -kl option 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 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 94. ♦ If the default realm for the Kerberos client on the database server machine 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:

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 are still 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 be considered an additional security measure for database access, since enabling integrated or Kerberos logins means that the database is relying on the security of the operating system on which it is running. If the database is shut down and copied to another computer (such as a user's computer) access to the database reverts to the SQL Anywhere security model and not the security model of the operating system of the computer where the database has been copied.

See also

- "Security concerns: Copied database files" on page 102
- "SET OPTION statement" [*SQL Anywhere Server - SQL Reference*]

Security concerns: Copied database files

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.

If a database contains sensitive information that needs to be protected from unauthorized access, the computer where the database files are stored should be protected from unauthorized access. If this is not possible, then there is a security risk since the database files could be copied and unauthorized access to the data may be obtained on another computer. To increase security in such environments, the following steps are recommended:

- User passwords, especially those with DBA authority, should be complex and difficult to guess.
- The PUBLIC.login_mode database option should be set 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 101.
- You should strongly encrypt the database file using the AES encryption algorithm. The encryption key should be complex and difficult to guess.

CHAPTER 4

Client/Server Communications

Contents

About this chapter

Each network environment has its own peculiarities. This chapter describes those aspects of network communication that are relevant to the proper functioning of your database server, and provides some tips for diagnosing network communication problems. It describes how networks operate, and provides hints on running the network database server under each protocol.

Network database server only
The material in this chapter applies only to the network server. You do not need to read this chapter if you are using the personal database server.

Supported network protocols

Properly configured SQL Anywhere database servers run under the following networks and protocols:

- **Windows XP/200x (except Windows 2003 Server)** TCP/IP or SPX protocols.
- **Windows 2003 Server (32-bit)** TCP/IP or SPX protocols.
- **Windows 2003 Server (64-bit)** TCP/IP protocol.
- **Windows CE** TCP/IP protocol.
- **NetWare** TCP/IP or SPX protocol.
- **Unix** TCP/IP protocol.

The client library for each platform supports the same protocols as the corresponding server. In order for SQL Anywhere to run properly, the network protocols (TCP/IP and/or SPX) must be installed and configured properly on both the client and server computers.

Using the TCP/IP protocol

TCP/IP is a suite of protocols that has gained widespread use with the expansion of the Internet and the world wide web.

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.

☞ For more information, see "Transport-Layer Security" on page 885.

IPv6 support in SQL Anywhere

On IPv6-enabled computers, the network database server listens by default on all IPv6 addresses, as well as all IPv4 addresses on the computer. IPv6 is supported on Windows XP/200x, Linux, Mac OS X, Solaris, AIX, and HP-UX.

In most cases, no changes are required to the 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 will probably have 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:

```
dbsrv10 –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);...
```

On Windows, 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. This affects values specified by the following protocol options:

- "Broadcast protocol option [BCAST]" on page 243
- "Host protocol option [IP]" on page 249
- "MyIP protocol option [ME]" on page 256

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. For example:

```
LINKS=tcpip(BROADCAST=ff02::1%2)
```

Note that `ff02::1` is the IPv6 link-local multicast address.

Using TCP/IP with Windows

The TCP/IP implementation for database servers on all Windows platforms uses Winsock 2.2. Clients on Windows CE 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.

☞ For more information, see "DLL protocol option" on page 247.

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.

☞ For more information, see "-p server option" on page 163, or "CommBufferSize connection parameter [CBSIZE]" on page 210.

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 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.

☞ For more information, see "ClientPort protocol option [CPORT]" on page 245.

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.

```
ENG=myeng;LINKS=tcpip
(ClientPort=5050-5060;HOST=myhost;PORT=2020;DoBroadcast=NONE)
```

See also

- "CommLinks connection parameter [LINKS]" on page 211
- "ClientPort protocol option [CPORT]" on page 245
- "ServerPort protocol option [PORT]" on page 258
- "Host protocol option [IP]" on page 249
- "DoBroadcast protocol option [DOBROAD]" on page 247

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.

  For more information, see "Host protocol option [IP]" on page 249.

- **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.

  For more information, see "DoBroadcast protocol option [DOBROAD]" on page 247.

- **MyIP parameter** You should set **MyIP=NONE** on the client side.

  For more information, see "MyIP protocol option [ME]" on page 256.

- **TIMEOUT parameter** Set the TIMEOUT (TO) protocol option to increase the time the client will wait while searching for a server.

☞ For more information, see the "Timeout protocol option [TO]" on page 260.

A typical CommLinks (LINKS) connection parameter may look as follows:

```
LINKS=tcpip(MyIP=NONE;DoBroadcast=DIRECT;HOST=server_ip)
```

Encrypting client/server communications over TCP/IP

By default, communication packets are not encrypted; this 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.

☞ For more information, see “Transport-Layer Security” on page 885.

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 CE), Unix, or NetWare 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.

To enable this feature, 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 \win32* 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 608. 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 NetWare and Unix. If this entry is missing on Windows, Windows looks for an LDAP server running on the local domain controller.

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 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 is required for the database server, and ignored on the client.

password The password for authdn. This 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.

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
```

When the 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 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.

Using the SPX protocol

SPX is a protocol from Novell. SQL Anywhere for NetWare, and Windows XP/200x can employ the SPX protocol. This section provides some tips for using SPX under different operating systems.

SPX is not supported on 64-bit operating systems.

Connecting via SPX

Some computers can use the NetWare bindery. These computers are NetWare servers or Windows XP/200x computers where the Client Service for NetWare is installed. A client application on one of these computers does not need to use broadcasts to connect to the server if the server to which it is connecting is also using the bindery.

Applications running on computers not using the bindery must connect using one of the following:

- **An explicit address** You can specify an explicit address using the HOST protocol option.

  For more information, see the "Host protocol option [IP]" on page 249.

- **Broadcast** If a server is not found in the bindery, the client will attempt to find it using a broadcast. This can be disabled with by setting the DoBroadcast (DOBROAD) protocol option to NONE (client-side) or NO (server-side).

  For more information, see the "DoBroadcast protocol option [DOBROAD]" on page 247.

Adjusting communication compression settings to improve performance

Enabling compression for one or all connections, as well as setting the minimum size at which packets are compressed, can significantly 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*], or "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.
- 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 connection parameter [COMP]" on page 213, "-pc server option" on page 164.

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 or not changing the compression threshold is beneficial.

☞ For more information see "CompressionThreshold connection parameter [COMPTH]" on page 214 and "-pt server option" on page 164.

♦ 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 connection parameter [COMP]" on page 213 and "-pc server option" on page 164.

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 connection parameter [COMPTH]" on page 214 and "-pt server option" on page 164.

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.

Use logging

Specifying the -z database server option displays diagnostic communication messages, and other messages to the Server Messages window for troubleshooting purposes. This can help you diagnose how a connection is failing and what connection parameters are used for the connection attempt, as well as identify what communication links are being used.

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.

☞ For more information about the -x option, see the "-x server option" on page 182.

☞ For information about the CommLinks (LINKS) connection parameter, see "CommLinks connection parameter [LINKS]" on page 211.

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 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 enter the following command at the command prompt 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:

```
c:> ping 191.72.109.12
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.

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. This process 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:

```
c:> 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 75 and "Troubleshooting server startup" on page 44.

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:

- 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.

  For more information, see "Connecting across a firewall" on page 106.
- 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, make sure the Search Network For Database Servers option on the Database tab of the Connect dialog is cleared. You can select this option 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 242.

Adjusting timeout values

If you are experiencing problems with connections unexpectedly terminating, consider adjusting either the liveness or the idle timeout values.

☞ For more information about liveness timeout values, see "LivenessTimeout connection parameter [LTO]" on page 231, and the "-tl server option" on page 175.

☞ For more information about connection timeouts, see the "Idle connection parameter" on page 227, and the "-ti server option" on page 175.

CHAPTER 5

The Database Server

Contents

About this chapter

This chapter describes the options available for the SQL Anywhere database server.

The SQL Anywhere database server

Function

Start a personal database server or network database server.

Syntax

{ **dbeng10** | **dbsrv10** }
[*server-options*] [*database-file* [*database-options*] ...]

NetWare syntax

[**load**] **dbsrv10** [*server-options*] [*database-file* [*database-options*] ...]

Server options

Server option	Description
@data	Read in options from a configuration file or environment variable. See "@data server option" on page 125.
-?	Display usage information. See "-? server option" on page 126.
-b	Run in bulk operations mode. See "-b server option" on page 127.
-c *size*	Set initial cache size. See "-c server option" on page 127.
-ca 0	Disable dynamic cache sizing [Windows, Unix]. See "-ca server option" on page 129.
-cc { **+** | **-** }	Collect information about database pages to be used for cache warming. See "-cc server option" on page 130.
-ch *size*	Set the cache size upper limit [Windows XP/200x, Unix]. See "-ch server option" on page 131.
-cl *size*	Set the cache size lower limit [Windows XP/200x, Unix]. See "-cl server option" on page 132.
-cm *size*	Specify the amount of address space allocated for an Address Windowing Extensions (AWE) cache [Windows XP/200x]. See "-cm server option" on page 133.
-cr { **+** | **-** }	Warm the cache with database pages. See "-cr server option" on page 133.
-cs	Display cache usage in Server Messages window. See "-cs server option" on page 134.
-cv { **+** | **-** }	Control the appearance of messages about cache warming in the Server Messages window. See "-cv server option" on page 134.
-cw	Enable use of Address Windowing Extensions on Windows XP/200x for setting the size of the database server cache. See "-cw server option" on page 135.

Server option	Description
-dt	Specify the directory where temporary files are stored. See "-dt server option" on page 138.
-ec *encryption-options*	Enable packet encryption [network server]. See "-ec server option" on page 139.
-ep	Prompt for encryption key. See "-ep server option" on page 142.
-fc	Specify the file name of a DLL containing the file system full callback function. See "-fc server option" on page 143.
-fips	Require the use of FIPS-approved algorithms for strong database and communication encryption. See "-fips server option" on page 144.
-ga	Automatically unload the database after the last connection closed. In addition, shut down after the last database is closed [not NetWare]. See "-ga server option" on page 145.
-gb *level*	Set database process priority class to *level* [Windows XP/200x]. See "-gb server option" on page 145.
-gc *num*	Set maximum checkpoint timeout period to *num* minutes. See "-gc server option" on page 146.
-gd *level*	Set database starting permission. See "-gd server option" on page 146.
-ge *size*	Set the stack size for threads that run external functions. See "-ge server option" on page 147.
-gf	Disable firing of triggers. See "-gf server option" on page 148.
-gk *level*	Set the permission required to stop the server. See "-gk server option" on page 148.
-gl *level*	Set the permission required to load or unload data. See "-gl server option" on page 148.
-gm *num*	Set the maximum number of connections. See "-gm server option" on page 149.
-gn *num*	Set the maximum number of tasks that the database server can execute concurrently. See "-gn server option" on page 149.
-gp *size*	Set the maximum page size to *size* bytes. See "-gp server option" on page 150.
-gr *minutes*	Set the maximum recovery time to *num* minutes. See "-gr server option" on page 151.
-gss *size*	Set the thread stack size to *size* bytes [not applicable to Windows]. See "-gss server option" on page 151.

Server option	Description
-gt *num*	Set 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 server option" on page 152.
-gtc *logical-processors-to-use*	Control the maximum processor concurrency that the database server allows. See "-gtc server option" on page 153.
-gu *level*	Set the permission level for utility commands: utility_db, all, none, or DBA. See "-gu server option" on page 154.
-gx *num*	Specify the number of operating system threads for the database server to use. See "-gx server option" on page 155.
-k	Control the collection of Performance Monitor statistics. See "-k server option" on page 155.
-kl *GSS-API-library-file*	Specify 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 server option" on page 156.
-kr *server-realm*	Specify the realm of the Kerberos server principal and enable Kerberos authenticated connections to the database server. See "-kr server option" on page 157.
-krb	Enable Kerberos-authenticated connections to the database server. See "-krb server option" on page 157.
-m	Truncate the transaction log after each checkpoint for all databases. See "-m server option" on page 158.
-n *name*	Use *name* as the name of the database server. Note that the -n option is positional. See "-n server option" on page 159.
-o *filename*	Output messages to the specified file. See "-o server option" on page 161.
-oe *filename*	Specify file to log startup errors, fatal errors and assertions to. See "-oe server option" on page 161.
-on *size*	Specify a maximum size for the console log, after which the file is renamed with the extension *.old* and a new file is started. See "-on server option" on page 161.
-os *size*	Limit the size of the log file for messages. See "-os server option" on page 162.
-ot *filename*	Truncate the console log and append output messages to it. See "-ot server option" on page 163.
-p *packet-size*	Set the maximum network packet size [network server]. See "-p server option" on page 163.
-pc	Compress all connections except same-computer connections. See "-pc server option" on page 164.

Server option	Description
-pt *size_in_bytes*	Set the minimum network packet size to compress. See "-pt server option" on page 164.
-qi	Do not display the database server tray icon or Server Messages window [Windows XP/200x]. See "-qi server option" on page 165.
-qn	Do not minimize the Server Messages window on startup. See "-qn server option" on page 165.
-qp	Suppress messages about performance in the Server Messages window. See "-qp server option" on page 166.
-qs	Suppress startup error dialogs. See "-qs server option" on page 166.
-qw	Do not display the database server screen. See "-qw server option" on page 167.
-r	Opens database in read-only mode. See "-r server option" on page 167.
-s *facility-ID*	Set the syslog facility ID [Unix]. See "-s server option" on page 168.
-sb { **0** | **1** }	Specify how the server reacts to broadcasts. See "-sb server option" on page 169.
-sf *feature-list*	Secure features for databases running on this database server. See "-sf server option" on page 170.
-sk *key*	Specify a key that can be used to enable features that are disabled for the database server. See "-sk server option" on page 173.
-su *password*	Set the password for the DBA user of the utility database (utility_db), or disable connections to the utility database. See "-su server option" on page 174.
-ti *minutes*	Client idle time before shutdown—default 240 minutes. See "-ti server option" on page 175.
-tl *seconds*	Default liveness timeout for clients in seconds—default 120 seconds. See "-tl server option" on page 175.
-tmf	Force transaction manager recovery for distributed transactions [Windows XP/200x]. See "-tmf server option" on page 176.
-tmt *milliseconds*	Set the reenlistment timeout for distributed transactions [Windows XP/200x]. See "-tmt server option" on page 177.
-tq *time*	Set quitting time [network server]. See "-tq server option" on page 177.
-u	Use buffered disk I/O [Windows XP/200x, Unix]. See "-u server option" on page 177.
-ua	Turn off use of asynchronous I/O [Linux]. See "-ua server option" on page 178.

Server option	Description
-uc	Start the database server in console mode [Unix]. See "-uc server option" on page 178.
-ud	Run as a daemon [Unix]. See "-ud server option" on page 179.
-uf	Specify the action to take when a fatal error occurs [Unix]. See "-uf server option" on page 179.
-ui	Open the Server Startup Options dialog and display the Server Messages window, or start the database server in console mode if a usable display isn't available [Linux]. See "-ui server option" on page 180.
-ut *minutes*	Touch temporary files every *min* minutes [Unix]. See "-ut server option" on page 181.
-ux	Display the Server Messages window and Server Startup Options dialog [Linux]. See "-ux server option" on page 181.
-v	Display database server version and stop. See "-v server option" on page 182.
-x *list*	Comma-separated list of communication links to use. See "-x server option" on page 182.
-xa *authentication-info*	Specify a list of database names and authentication strings for an arbiter server. See "-xa server option" on page 184.
-xf *state-file*	Specify the location of the file used for maintaining state information about your database mirroring system. See "-xf server option" on page 184.
-xs	Specify server side web services communications protocols. See "-xs server option" on page 185.
-z	Provide diagnostic information on communication links [network server]. See "-z server option" on page 187.
-ze	Display database server environment variables in the Server Messages window. See "-ze server option" on page 187.
-zl	Turn on capturing of the most recently-prepared SQL statement for each connection. See "-zl server option" on page 187.
-zn	Specifies the number of request log file copies to retain. See "-zn server option" on page 188.
-zo *filename*	Redirect request logging information to a separate file. See "-zo server option" on page 189.
-zp	Turn on capturing of the plan most recently used by the query optimizer. See "-zp server option" on page 190.
-zr { **all** | **SQL** | **none** }	Turn on logging of SQL operations. The default is NONE. See "-zr server option" on page 190.

Server option	Description
-zs *size*	Limit the size of the log file used for request logging. See "-zs server option" on page 192.
-zt	Turn on logging of request timing information. See "-zt server option" on page 192.

Recovery options

Recovery option	Description
-f	Force the database to start without a transaction log. See "-f recovery option" on page 193.

Database options

Database option	Description
-a *filename*	Apply the named transaction log file. See "-a database option" on page 194.
-ad *log-directory*	Specify the directory containing log files to be applied to the database. See "-ad database option" on page 195.
-ar	Apply any log files located in the same directory as the transaction log to the database. See "-ar database option" on page 195.
-as	Continue running the database after transaction logs have been applied (used in conjunction with -ad or -ar). See "-as database option" on page 196.
-dh	Do not display this database when dblocate is used against this server. Note that the -dh option is positional. See "-dh database option" on page 197.
-ek *key*	Specify encryption key. See "-ek database option" on page 197.
-m	Truncate (delete) the transaction log after each checkpoint for the specified database. See "-m database option" on page 198.
-n *name*	Name the database. Note that the -n option is positional. See "-n database option" on page 199.
-r	Open the specified database(s) in read-only mode. Database modifications not allowed. See "-r database option" on page 199.
-sn *alternate-server-name*	Provide an alternate server name for a single database running on a database server. See "-sn database option" on page 200.
-xp *mirroring-options*	Provide 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 database option" on page 202.

Description

The **dbeng10** command starts a personal database server. The **dbsrv10** command starts a network database server.

Cache size

The amount of cache memory available to the database server can be a key factor in affecting performance. The database server takes an initial amount of cache memory that is either specified by the -c option or is a default value.

☞ For information on the default cache size, see "-c server option" on page 127.

On Windows XP/200x and Unix, the database server automatically takes more memory for use in the cache as needed, as determined by a heuristic algorithm.

☞ For more information, see "Use the cache to improve performance" [*SQL Anywhere Server - SQL Usage*].

You can use database options to configure the upper cache limit: see "-ch server option" on page 131. As well, you can force the cache to remain at its initial amount: see "-ca server option" on page 129.

Server differences

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.

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.

Platform availability

Both personal and network database servers are supplied for each supported operating system, with the following exceptions:

- **Novell NetWare** Only the network server is supplied.
- **Windows CE** Only the network 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.

NetWare notes

You can only run one database server for each major version of SQL Anywhere on NetWare. Attempting to run a second database server using the same major version of the software fails because of conflicts in exported functions and shared memory over SPX.

On NetWare, the database file and the transaction log file must be on a NetWare volume, and the paths must be fully specified. NetWare allows you to have volumes that span two or more hard disks.

Database file 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.

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\10.0*.

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 zero, no messages appear at all. A better setting would be 1, so that informational and warning messages do not appear, but errors do. The default setting (no entry present) is for all message types to appear.

See also

- "Running the Database Server" on page 11
- "Network protocol options" on page 242

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 and -qs 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 dialogs caused by the -v (version) and -ep (prompt for database encryption password) server options.

Database server options

These options apply to the server as a whole, not just to an individual database.

@data server option

Function

Read in options from the specified environment variable or configuration file

Syntax

{ **dbsrv10** | **dbeng10** } *@data* …

Applies to

All operating systems and servers, as well as all database utilities except the Language Selection utility (dblang), the Rebuild utility (rebuild), the Certificate Generation utility (gencert), the Certificate Reader utility (readcert), the ActiveSync provider install utility (mlasinst), and the File Hiding utility (dbfhide).

Description

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.

☞ For more information about using configuration files, see "Using configuration files" on page 587.

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.

☞ For more information about the File Hiding utility, see "The File Hiding utility" on page 608.

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.

Examples

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:

```
dbsrv10 @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, entered all on one line, 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 statement starts the database server using an environment variable named **envvar**.

```
dbsrv10 @envvar
```

-? server option

Function

Display usage information.

Syntax

{ **dbsrv10** | **dbeng10** } **-?**

Applies to

All operating systems and servers.

Description

Displays a short description of each server option. The database doesn't perform any other task.

-b server option

Function

Use bulk operation mode.

Syntax

{ **dbsrv10** | **dbeng10** } **-b** …

Applies to

All operating systems and servers.

Description

This is useful for using the Interactive SQL INPUT command to load large quantities of data into a database.

The -b option should not be used if you are using LOAD TABLE to bulk load data.

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 log file.

Bulk operation mode doesn't disable the firing of triggers.

-c server option

Function

Set the initial memory reserved for caching database pages and other server information.

Syntax

{ **dbsrv10** | **dbeng10** } **-c** { *size*[**k** | **m** | **g** | **p**] } …

Applies to

All operating systems and servers.

Description

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 the 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 physical system memory, or of the available address space, whichever is lower. Available address space depends on the operating system. For example:

- 2.8 GB for Windows 32-bit Advanced Server, Enterprise Server and Datacenter Server
- 3.8 GB for the 32-bit database server running on Windows x64 Edition
- 1.8 GB on all other 32-bit systems
- On Windows CE, the available address space is a percentage of the available physical system memory

If you use **p**, the argument is a percentage. You can use % as an alternative to p, but as most non-Unix operating systems use % 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:

```
dbeng10 -c 50%% ...
```

On NetWare and Unix operating systems, the cache size is set to the lesser of:

- the value specified after -c
- `95% of (`*available memory*` - 5 MB)`

On Windows CE, the cache size will be set to the lesser of:

- the value specified after -c
- `95% of (`*available memory*` - 2 MB)`

If no -c option is provided, the database server computes the initial cache allocation as follows:

1. It uses the following operating-system-specific default cache sizes:
   - **Windows CE** 600 KB
   - **Windows XP/200x, NetWare** 2 MB
   - **Unix** 8 MB
2. It computes a runtime-specific minimum default cache size, which is the lesser of the following items:
   - 25% of the computer's physical memory

- 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. It allocates the greater of the two values computed.

NetWare database server
There is a tradeoff between memory for the database server and memory for the NetWare file system buffers. A larger database server cache will improve database server performance at the expense of NetWare file system performance. If the database server cache is too big, NetWare will report an error that there is insufficient memory for cache buffers.
NetWare memory requirements increase with every new directory and file on the file server. To track memory usage on the NetWare server, load *monitor.nlm* (if it isn't already loaded) and select "Resource Utilization". Extra memory for your NetWare server computer could improve database performance and/or file server performance dramatically.

If your database is encrypted, you may want to increase the cache size. As well, if you are using dynamic cache resizing (-ca option), then the cache size that is used may be restricted by the amount of memory that is available.

☞ For more information about setting the cache size, see "Increase the cache size" [*SQL Anywhere Server - SQL Usage*].

The 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( 'CacheSize' )
```

See also

- "-ca server option" on page 129
- "-ch server option" on page 131
- "-cl server option" on page 132
- "Limiting the memory used by the cache" [*SQL Anywhere Server - SQL Usage*]

Example

The following example, entered all on one line, starts a server named **myserver** that starts with a cache size of 3 MB and loads the sample database:

```
dbeng10 -c 3m -n myserver "samples-dir\demo.db"
```

☞ For information about the default location of *samples-dir*, see "The samples directory" on page 295.

-ca server option

Function

Enforce a static cache size.

Syntax

{ **dbsrv10** | **dbeng10** } **-ca 0** …

Applies to

Windows, Unix

Description

You can disable automatic cache increase because of high server load 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. The cache size still increases if the database server would otherwise run into the error `Fatal Error: dynamic memory exhausted`.

This server option must only be used in the form -ca 0.

See also

- "-c server option" on page 127
- "-ch server option" on page 131
- "-cl server option" on page 132
- "Limiting the memory used by the cache" [*SQL Anywhere Server - SQL Usage*]

Example

The following example starts a server named **myserver** that has a static cache that is 40% of the available physical memory and loads the sample database:

```
dbsrv10 -c 40P -ca 0 -n myserver "samples-dir\demo.db"
```

☞ For information about the default location of *samples-dir*, see "The samples directory" on page 295.

-cc server option

Function

Collect information about database pages to be used for cache warming the next time the database is started.

Syntax

{ **dbsrv10** | **dbeng10** } **-cc** { **+** | **-** } ...

Applies to

All operating systems and servers.

Description

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. Note that 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

- "-cr server option" on page 133
- "-cv server option" on page 134
- "Using cache warming" [*SQL Anywhere Server - SQL Usage*]

-ch server option

Function

Sets a maximum cache size, as a limit to automatic cache growth.

Syntax

{ **dbsrv10** | **dbeng10** } **-ch** { *size*[**k** | **m** | **g** | **p**] } ...

Applies to

Windows, Unix

Description

This option limits the size of the database server cache during automatic cache growth. By default the upper limit is approximately the lower of available address space and 90% of the 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 available address space, whichever is lower. Available address space depends on the operating system. For example:

- 2.8 GB for Windows 32-bit Advanced Server, Enterprise Server and Datacenter Server
- 3.8 GB for the 32-bit database server running on Windows x64 Edition
- 1.8 GB on all other 32-bit systems
- On Windows CE, the available address space is a percentage of the available physical system memory

If you use **p**, the argument is a percentage. You can use % as an alternative to P, but as most non-Unix operating systems use % 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:

```
dbeng10 -ch 50%% ...
```

See also

- "-c server option" on page 127
- "-ca server option" on page 129
- "-cl server option" on page 132
- "Limiting the memory used by the cache" [*SQL Anywhere Server - SQL Usage*]

Example

The following example starts a server named silver that has a maximum cache size of 2 MB and loads the sample database:

```
dbeng10 -ch 2m -n silver "samples-dir\demo.db"
```

☞ For information about the default location of *samples-dir*, see "The samples directory" on page 295.

-cl server option

Function

Set a minimum cache size as a lower limit to automatic cache resizing.

Syntax

{ **dbsrv10** | **dbeng10** } **-cl** { *size*[**k** | **m** | **g** | **p**] } ...

Applies to

Windows, Unix

Description

This option sets a lower limit to the cache. The default minimum cache size is the initial cache size.

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 available address space, whichever is lower. Available address space depends on the operating system. For example:

- 2.8 GB for Windows 32-bit Advanced Server, Enterprise Server and Datacenter Server
- 3.8 GB for the 32-bit database server running on Windows x64 Edition
- 1.8 GB on all other 32-bit systems
- On Windows CE, the available address space is a percentage of the available physical system memory

If you use **p**, the argument is a percentage. You can use % as an alternative to P, but as most non-Unix operating systems use % 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:

```
dbeng10 -cl 50%% ...
```

See also

- "-c server option" on page 127
- "-ca server option" on page 129
- "-ch server option" on page 131
- "Limiting the memory used by the cache" [*SQL Anywhere Server - SQL Usage*]

Example

The following example starts a server named silver that has a minimum cache size of 5 MB and loads the database file *example.db*:

```
dbeng10 -cl 5m -n silver "c:\example.db"
```

-cm server option

Function

Specify the amount of address space allocated for an Address Windowing Extensions (AWE) cache on Windows XP/200x.

Syntax

{ **dbsrv10** | **dbeng10** } **-cm** { *size*[**k** | **m** | **g** | **p**] } …

Applies to

Windows XP/200x

Description

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 sufficient. 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 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 available address space. If you use **p**, the argument is a percentage. You can use % as an alternative to P, but as most non-Unix operating systems use % as an environment variable escape character, you must escape the % character. To set the minimum cache size to 50 percent of the address space, you would use the following:

```
dbeng10 -cm 50%% ...
```

See also

- "-ca server option" on page 129
- "-ch server option" on page 131
- "-cl server option" on page 132
- "-cw server option" on page 135
- "Limiting the memory used by the cache" [*SQL Anywhere Server - SQL Usage*]

-cr server option

Function

Reload (warm) the cache with database pages using information collected the last time the database was run.

Syntax

{ **dbsrv10** | **dbeng10** } **-cr** { **+** | **-** } ...

Applies to

All operating systems and servers.

Description

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 server option" on page 130
- "-cv server option" on page 134
- "Using cache warming" [*SQL Anywhere Server - SQL Usage*]

-cs server option

Function

Display cache size changes in the Server Messages window.

Syntax

{ **dbsrv10** | **dbeng10** } **-cs** ...

Applies to

Windows, Unix

Description

For troubleshooting purposes, display cache information in the Server Messages window whenever the cache size changes.

-cv server option

Function

Control the appearance of messages about cache warming in the Server Messages window.

Syntax

{ **dbsrv10** | **dbeng10** } **-cv** { **+** | **-** } ...

Applies to

All operating systems and servers.

Description

When -cv+ is specified, a message appears in the 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)

By default, this option is turned off.

See also

- "-cc server option" on page 130
- "-cr server option" on page 133
- "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 Server Messages window:

```
dbsrv10 -cc+ -cr+ -cv+ mydatabase.db
```

-cw server option

Function

Enable use of Address Windowing Extensions (AWE) on Windows XP/200x for setting the size of the database server cache.

Syntax

{ **dbsrv10** | **dbeng10** } **-cw** ...

Applies to

Windows XP/200x

Description

The amount of memory available for use as a database server cache is one of the key factors controlling performance. Because Windows XP/200x 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.

Operating system	Maximum non-AWE cache size	Maximum amount of physical memory supported by Windows
Windows 2000 Professional	1.8 GB	4 GB
Windows 2000 Server	1.8 GB	4 GB
Windows 2000 Advanced Server	2.7 GB*	8 GB
Windows 2000 Datacenter Server	2.7 GB*	64 GB
Windows XP Home Edition	1.8 GB	2 GB
Windows XP Professional	1.8 GB	4 GB
Windows Server 2003, Web Edition	1.8 GB	2 GB
Windows Server 2003, Standard Edition	1.8 GB	4 GB
Windows Server 2003, Enterprise Edition	2.7 GB*	32 GB
Windows Server 2003, Datacenter Edition	2.7 GB*	64 GB

*You must start the operating system using the `/3GB` option to use a cache of this size.

When using an AWE cache, almost all of the available physical memory in the system can be allocated for the cache.

If you can set a cache of the desired 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). While this is sufficient in most cases, you can change the amount of reserved address space using the -cm option.

☞ For information about specifying the cache size, see "-cm server option" on page 133.

To start a database server with an AWE cache, you must do the following:

- Have at least 130 MB of memory available on your system.
- If your system has between 2 GB and 16 GB of memory, add the `/3GB` option to the Windows boot line in the "[operating systems]" section of the *boot.ini* file.

If your system has more than 16 GB of memory, do not add the `/3GB` option to the Windows boot line in the "[operating systems]" section of the *boot.ini* file because Windows won't be able to address memory beyond 16 GB.

- If your system has more than 4 GB of memory, add the `/PAE` option to the Windows boot line in the "[operating systems]" section of the *boot.ini* file.
- 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 2000.

  1. Log on to Windows as Administrator.
  2. From the Start menu, choose Settings ► Control Panel.
  3. Open the Administrative Tools folder.
  4. Double-click Local Security Policy.
  5. Open Local Policies in the left pane.
  6. Double-click User Rights Assignment in the left pane.
  7. Double-click the Lock Pages In Memory policy in the right pane.

     The Local Security Policy Setting dialog appears.
  8. In the Local Security Policy Setting dialog, click Add.

     The Select Users or Groups dialog appears.
  9. Select the user ID from the list and click Add.
  10. In the Local Security Policy Setting dialog, click OK.
  11. 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. On 64-bit Windows platforms, the cache page size is at least 8 KB.

☞ For more information about dynamic cache sizing, see “Use the cache to improve performance” [*SQL Anywhere Server - SQL Usage*].

See also

- “-c server option” on page 127
- “-ch server option” on page 131
- “-cm server option” on page 133

Example

The following example starts a server named **myserver** that starts with a cache size of 12 GB and loads the database *c:\test\mydemo.db*:

```
dbeng10 -c 12G -cw c:\test\mydemo.db
```

-dt server option

Function

Specifies the directory where temporary files are stored.

Syntax

{ **dbsrv10** | **dbeng10** } **-dt** *temp-file-dir* ...

Applies to

All servers and operating systems, except shared memory connections on Unix.

Description

You can use the -dt option to specify a directory for 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 shared memory connections 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

- “Overview of database files” on page 266
- “SATMP environment variable” on page 285
- “Place different files on different devices” [*SQL Anywhere Server - SQL Usage*]

-ec server option

Function

Use 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

{ **dbsrv10** | **dbeng10** } **-ec** *encryption-options* …

encryption-options :

{ **NONE** |
SIMPLE |
TLS (TLS_TYPE={**ECC** | **RSA**};
   CERTIFICATE=*filename*;**CERTIFICATE_PASSWORD=***password*
   [**FIPS=**{ **ON** | **OFF** };] **)**

}, …

Description

You can use this option to secure communication packets between client applications and the database server using transport-layer security.

☞ For more information, see “Transport-Layer Security” on page 885.

Separately licensed component required
ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.
See “Separately licensed components” [*SQL Anywhere 10 - Introduction*].

The -ec option instructs the database server to accept *only* connections that are encrypted using one of the specified types. Connections over the TDS protocol, which include Java applications using jConnect, are always accepted regardless of the usage of the -ec option, and are never encrypted. Setting the TDS protocol option to NO disallows these unencrypted TDS connections. See “TDS protocol option” on page 260.

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. The -ec option controls the server's encryption settings and requires at least one of the following parameters in a comma-separated list:

- **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, as well as 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:

  - **TLS_TYPE** the type of encryption.

    This is either **ECC** or **RSA**.

    ECC specifies connections that are encrypted using the elliptic curve-based Certicom encryption technology. ECC encryption is not available on all platforms. For a list of supported platforms, see the Separately licensed components table in SQL Anywhere 10.0.0 Components by Platform. To use this type of encryption, the connection must be over the TCP/IP port.

    If you are using ECC encryption, you must generate ECC certificates. See "Creating digital certificates" on page 890.

    Note
    Version 10 clients cannot connect to version 9.0.2 or earlier database servers using the ECC algorithm. If you require strong encryption for this configuration, you can use the RSA algorithm.

    RSA specifies connections that are encrypted using RSA-based encryption technology. RSA encryption is not available on all platforms. For a list of supported platforms, see the Separately licensed components table in SQL Anywhere 10.0.0 Components by Platform. To use this type of encryption, the connection must be over the TCP/IP port.

    If you are using RSA encryption, you must generate RSA certificates. See "Creating digital certificates" on page 890.

  - **CERTIFICATE** the file name of the server certificate.

    This is the server certificate containing the server's private key. This server certificate can be self-signed or signed by an enterprise root certificate or Certificate Authority.

  - **CERTIFICATE_PASSWORD** the password for the server certificate's private key.

  The TLS parameter accepts the following optional argument:

  - **FIPS** accepts connections that are encrypted using FIPS-approved RSA encryption technology. To use FIPS, the tls_type must be RSA. FIPS uses a separate approved library, but is compatible with clients specifying RSA with Adaptive Server Anywhere 9.0.2 or later. FIPS-approved RSA encryption is not available on all platforms. For a list of supported platforms, see the Separately licensed components table in SQL Anywhere 10.0.0 Components by Platform.

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 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 FIPS connections fail, and connections requesting no encryption use simple encryption.

The *dbecc10.dll* and *dbrsa10.dll* files contain the ECC and RSA code used for encryption and decryption. The file *dbfips10.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 on the 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 898
- "Encryption connection parameter [ENC]" on page 222
- "-ek database option" on page 197
- "-ep server option" on page 142
- "DatabaseKey connection parameter [DBKEY]" on page 217

Examples

The following example specifies that connections with no encryption and simple encryption are allowed.

```
dbsrv10 -ec NONE,SIMPLE -x tcpip c:\mydemo.db
```

The following example specifies starts a database server that uses the elliptic-curve server certificate *sample.crt*.

```
dbsrv10 -ec TLS
(TLS_TYPE=ECC;CERTIFICATE=sample.crt;CERTIFICATE_PASSWORD=tJ1#m6+W) -x tcpip
c:\mydemo.db
```

The following example starts a database server that uses the RSA server certificate *rsaserver.crt*.

```
dbsrv10 -ec TLS
(TLS_TYPE=RSA;CERTIFICATE=rsaserver.crt;CERTIFICATE_PASSWORD=test) -x tcpip
c:\mydemo.db
```

The following example starts a database server that uses the FIPS-approved RSA server certificate *rsaserver.crt*.

```
dbsrv10 -ec TLS
(TLS_TYPE=RSA;FIPS=ON;CERTIFICATE=rsaserver.crt;CERTIFICATE_PASSWORD=test) -
x tcpip c:\mydemo.db
```

-ep server option

Function

Prompt the user for the encryption key upon starting a strongly encrypted database.

Syntax

{ **dbsrv10** | **dbeng10** } **-ep** …

Description

The -ep option instructs the database server to make a dialog box 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 only if *all* of the following are 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 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 885.

See also

- “Starting the database server with transport-layer security” on page 898
- “-ec server option” on page 139
- “Encryption connection parameter [ENC]” on page 222
- “-ek database option” on page 197
- “DatabaseKey connection parameter [DBKEY]” on page 217

Example

The user is prompted for the encryption key when the *myencrypted.db* database is started:

```
dbsrv10 -ep -x tcpip myencrypted.db
```

-fc server option

Function

Specify the file name of a DLL (or shared object on Unix) containing the File System Full callback function.

Syntax

{ **dbsrv10** | **dbeng10** } **-fc** *filename* …

Applies to

All operating systems and servers.

Description

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 the default location of *samples-dir*, see “The samples directory” on page 295.

SQL Anywhere searches for the callback function DLL in the same locations as it searches for other DLLs and files.

☞ For information about where SQL Anywhere searches for files, see “How SQL Anywhere locates files” on page 296.

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 Windows platforms, if the database server has been started with a Server Messages window (neither -qi nor -qw have been specified), and a callback DLL has not been provided, a dialog appears when a disk full condition occurs. This dialog 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 in case of a disk space shortage.

☞ For more information, see “CREATE EVENT statement” [*SQL Anywhere Server - SQL Reference*].

See also

- “Using callback functions” [*SQL Anywhere Server - Programming*]
- “Understanding system events” on page 815
- “max_temp_space option [database]” on page 431
- “temp_space_limit_check option [database]” on page 461

Example

When the database server starts, it attempts to load the *diskfull.dll* DLL.

```
dbeng10 -fc diskfull.dll
```

-fips server option

Function

Requires that only FIPS-approved algorithms should be used for strong database and communication encryption.

Syntax

{ **dbsrv10** | **dbeng10** } **-fips** …

Description

Specifying this option forces all server encryption to use FIPS-approved algorithms. This option applies to strong database encryption, client/server transport-layer security, and web services transport-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-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.
See “Separately licensed components” [*SQL Anywhere 10 - Introduction*].

For strong database encryption, the -fips option causes new databases to use the AES_FIPS type, even if AES is 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 or AES_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.

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 873
- "Transport-Layer Security" on page 885
- "Using transport-layer security for SQL Anywhere web services" on page 902
- "-ec server option" on page 139
- "ENCRYPT function [String]" [*SQL Anywhere Server - SQL Reference*]
- "HASH function [String]" [*SQL Anywhere Server - SQL Reference*]

-ga server option

Function

Unload database after last connection is dropped.

Syntax

{ **dbsrv10** | **dbeng10** } **-ga** …

Applies to

All operating systems except NetWare.

Description

Specifying this option on the network server causes each database to be unloaded after the last connection to it is dropped. In addition to unloading each database after the last connection is dropped, the server shuts down when the last database is stopped.

-gb server option

Function

Set the server process priority class.

Syntax

{ **dbsrv10** | **dbeng10** } **-gb** { **idle** | **normal** | **high** | **maximum** } …

Applies to

Windows XP/200x

Description

Set the server process priority class. The value idle is provided for completeness, and maximum may interfere with the running of your computer. Normal and high are the commonly-used settings.

-gc server option

Function

Set maximum desired interval between checkpoints.

Syntax

{ **dbsrv10** | **dbeng10** } **-gc** *integer* …

Applies to

All operating systems and servers.

Description

Set the *maximum* desired length of time in minutes that the database server runs without doing a checkpoint on each database.

The default value is 60 minutes. If a value of 0 is entered, the default value of 60 minutes is used.

Checkpoints generally occur more frequently than the specified time.

☞ For more information, see "How the database server decides when to checkpoint" on page 786.

See also

- "checkpoint_time option [database]" on page 399
- "Checkpoints and the checkpoint log" on page 783

-gd server option

Function

Set permissions required to start or stop a database.

Syntax

{ **dbsrv10** | **dbeng10** } **-gd** { **DBA** | **all** | **none** } …

Applies to

All operating systems and servers.

Description

This is the permission required for a user to cause a new database file to be loaded by the server, or to stop a database on a running database server. The level can be one of the following:

- **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 apart from when the database server itself is started and stopped.

The default setting is all for the personal database server, and DBA for the network database server. Both uppercase and lowercase syntax is acceptable.

Note that when this option is set to DBA, the client application must already have a connection to the server to start or stop a database. Providing a DBA user ID and password on a new connection isn't sufficient.

You can obtain the setting of the -gd option using the StartDBPermission server property:

```
SELECT PROPERTY ( 'StartDBPermission' )
```

Example

The following steps illustrates how to use the -gd option for the network database server.

1. Start the network database server:

   ```
   dbsrv10 -x tcpip -su mypwd -n myserver -gd DBA
   ```

2. Connect to the utility database from Interactive SQL:

   ```
   dbisql -c "UID=DBA;PWD=mypwd;ENG=myserver;DBN=utility_db"
   ```

3. Start a database:

   ```
   START DATABASE demo
   ON myserver;
   ```

4. Connect to the database you have started:

   ```
   CONNECT
   TO myserver
   DATABASE demo
   USER DBA IDENTIFIED BY sql
   ```

-ge server option

Function

Set stack size for external functions.

Syntax

{ **dbsrv10** | **dbeng10** } **-ge** *integer* …

Applies to

Windows XP/200x, NetWare

Description

Sets the stack size for threads running external functions, in bytes. The default is 32 KB.

-gf server option

Function

Disable firing of triggers by the server.

Syntax

{ **dbsrv10** | **dbeng10** } **-gf** …

Applies to

All operating systems and servers.

Description

The -gf server option instructs the server to disable the firing of triggers.

See also

- "fire_triggers option [compatibility]" on page 414

-gk server option

Function

Set the permission required to stop the network server and personal server using dbstop.

Syntax

{ **dbsrv10** | **dbeng10** } **-gk** { **DBA** | **all** | **none** } …

Applies to

All operating systems and servers.

Description

The allowed values include:

- **DBA** Only users with DBA authority can use dbstop to stop the server. This is the default for the network server.
- **all** All users can use dbstop to stop the server. This is the default for the personal server.
- **none** The server cannot be stopped using dbstop.

Both uppercase and lowercase syntax is acceptable.

-gl server option

Function

Set the permission required to load data using LOAD TABLE, and to unload data using UNLOAD or UNLOAD TABLE.

Syntax

{ **dbsrv10** | **dbeng10** } **-gl** { **DBA** | **all** | **none** } …

Applies to

All operating systems and servers.

Description

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.

The allowed values are as follows:

- **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.

Both uppercase and lowercase syntax is acceptable.

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. These settings reflect the fact that, on non-Unix platforms, the personal database server is running on the current computer, and so the user already has access to the file system.

-gm server option

Function

Limit the number of concurrent connections to the server.

Syntax

{ **dbsrv10** | **dbeng10** } **-gm** *integer* …

Applies to

All operating systems and servers.

Description

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.

The database server allows one extra DBA connection above the connection limit to allow a DBA to connect to the server and drop other connections in an emergency.

-gn server option

Function

Set the maximum number of tasks that the database server can execute concurrently.

Syntax

{ **dbsrv10** | **dbeng10** } **-gn** *integer* …

Applies to

All operating systems and servers.

Description

This option sets the maximum multiprogramming level of the database server. It limits the 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 maximum number of combined unscheduled and active requests is limited by the -gm server option, which limits the number of connections to the server.

The default value is 20 active tasks for both the network database server and the personal database server, except on Windows CE where the default is 3, and 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 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 is if the request involves the execution of a Java stored procedure or function 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. Consequently, these tasks count toward the -gn limit as if they were actually 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.

See also

- "Threading in SQL Anywhere" on page 22
- "Setting the database server's multiprogramming level" on page 25
- "max_query_tasks option [database]" on page 428
- "-gm server option" on page 149
- "-gm server option" on page 149
- "-gtc server option" on page 153
- "-gx server option" on page 155

-gp server option

Function

Set the maximum allowed database page size.

Syntax

{ **dbsrv10** | **dbeng10** } **-gp** { **2048** | **4096** | **8192** | **16384** | **32768** } …

Applies to

All operating systems and servers.

Description

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.

If you do not use this option, then the page size of the first database on the command line is used.

On all platforms, if you do not use this option and start a server with no databases loaded, the default value is 4096.

-gr server option

Function

Set the maximum length of time (in minutes) for recovery from system failure.

Syntax

{ **dbsrv10** | **dbeng10** } **-gr** *integer* …

Applies to

All operating systems and servers.

Description

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.

☞ For more information, see "recovery_time option [database]" on page 449.

-gss server option

Function

Set the stack size per internal execution thread in the server.

Syntax

{ **dbsrv10** | **dbeng10** } **-gss** { *integer*[**k** | **m**] } …

Applies to

This option has no effect on Windows operating systems.

Description

The number of internal execution threads is controlled by the -gn option, and has a default value of 20. 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.

On NetWare, the default, and minimum, stack size per internal execution thread is 128 KB, and the maximum stack size is 1 MB. On Unix, the default and minimum stack size per internal execution thread is 500 KB, and the maximum stack size is 4 MB.

See also

- "Threading in SQL Anywhere" on page 22

-gt server option

Function

Set the maximum number of physical processors that can be used (up to the licensed maximum). This option is only useful on multiprocessor systems.

Syntax

{ **dbsrv10** | **dbeng10** } **-gt** *integer* …

Applies to

Windows (except Windows CE), Linux, and Solaris.

Description

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. In addition, the personal database server and runtime database server are both limited to a single processor.

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 will, by default, run on all logical processors (hyperthreads and cores) of *n* physical processors. This can be further restricted with the -gtc option.

Valid values for the -gt option are 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 (dbeng10) the server uses a -gt value of 1.

See also

- "-gn server option" on page 149
- "-gtc server option" on page 153
- "-gx server option" on page 155
- "Threading in SQL Anywhere" on page 22

-gtc server option

Function

Control the maximum processor concurrency that the database server allows.

Syntax

{ **dbsrv10** | **dbeng10** } **-gtc** *logical-processors-to-use* …

Applies to

Linux, Solaris, and Windows operating systems executing on Intel-compatible x86 and x64 platforms, excluding Windows CE.

Description

When you start the database server, the number of physical and logical processors detected by the database server appears in the 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. On Windows platforms, the number of threads created can be changed using the -gx option.

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 (see "-gt server option" on page 152), 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 [database]" on page 428.

See also

- "-gn server option" on page 149
- "-gt server option" on page 152
- "-gx server option" on page 155

- “Parallelism during query execution” [*SQL Anywhere Server - SQL Usage*]
- “Threading in SQL Anywhere” on page 22

Examples

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 (hence 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 ♦ -gn 20 ♦ -gx 21 Threads can execute on A0 and A1.
No licensing restrictions on the CPU with -gtc 5 specified	♦ -gt 4 ♦ -gtc 5 ♦ -gn 20 ♦ -gx 21 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 ♦ -gn 20 ♦ -gx 21 Threads can execute on A0, A1, B0, B1, and C0.
No licensing restrictions on the CPU with -gtc 1 specified	♦ -gt 4 ♦ -gtc 1 ♦ -gn 20 ♦ -gx 2 Threads can execute only on A0.

-gu server option

Function

Set permission levels for utility commands.

Syntax

{ **dbsrv10** | **dbeng10** } **-gu** { **all** | **none** | **DBA** | **utility_db** } ...

Applies to

All operating systems and servers.

Description

Sets permission levels for utility commands such as CREATE DATABASE and DROP DATABASE. The level can be set to one of the following: utility_db, all, none, or DBA. The default is DBA.

The utility_db level restricts the use of these commands to only those users who can connect to the utility database. The all, none, and DBA levels permit all users, no users, or users with DBA authority, respectively, to execute utility commands.

-gx server option

Function

Specify the number of operating system threads for the database server to use.

Syntax

{ **dbsrv10** | **dbeng10** } **-gx** *integer*...

Applies to

Windows XP/200x only.

Description

This option is available only on Windows XP/200x. It is unavailable on all other platforms. By default, on Windows platforms the number of operating system threads that are created to execute server tasks is one greater than the value of the -gtc option. Specifying the -gx option overrides this default.

Permitted values are in the range specified by the -gtc option (lower limit) and the -gn option (upper limit). If the integer value specified by the -gx option is outside this range, the lower or upper limit is imposed.

Increasing -gx causes a reduction in address space (not physical memory) available for the cache. Moreover, by increasing the number of threads in use, the Windows operating system has responsibility for the scheduling of database server threads and this may have an impact on throughput and performance. This option should be used only on the advice of iAnywhere Solutions technical support.

See also

- ♦ "-gn server option" on page 149
- ♦ "-gt server option" on page 152
- ♦ "-gtc server option" on page 153
- ♦ "Threading in SQL Anywhere" on page 22

-k server option

Function

Controls the collection of Performance Monitor statistics.

Syntax

{ **dbsrv10** | **dbeng10** } **-k** ...

Applies to

All operating systems and servers.

Description

The database server collects Performance Monitor statistics by default.

If you specify -k when you start the database server, then the 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

- "Monitoring statistics using Sybase Central Performance Monitor" [*SQL Anywhere Server - SQL Usage*]

-kl server option

Function

Specify the file name of the Kerberos GSS-API library (or shared object on Unix) and enable Kerberos authenticated connections to the database server.

Syntax

{ **dbsrv10** | **dbeng10** } **-kl** *GSS-API-library-file* ...

Applies to

All operating systems except NetWare and Windows CE.

Description

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 server option" on page 157
- "-krb server option" on page 157
- "Kerberos connection parameter [KRB]" on page 229
- "Using Kerberos authentication" on page 93

- "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.

```
dbsrv10 -kl libgssapi_krb5.so -n my_server_princ /opt/myapp/kerberos.db
```

-kr server option

Function

Specify the realm of the Kerberos server principal and enable Kerberos authenticated connections to the database server.

Syntax

{ **dbsrv10** | **dbeng10** } **-kr** *server-realm* ...

Applies to

All operating systems except NetWare and Windows CE.

Description

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.

See also

- "-kl server option" on page 156
- "-krb server option" on page 157
- "Kerberos connection parameter [KRB]" on page 229
- "Using Kerberos authentication" on page 93
- "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.

```
dbeng10 -kr MYREALM -n my_server_princ C:\kerberos.db
```

-krb server option

Function

Enable Kerberos-authenticated connections to the database server.

Syntax

{ **dbsrv10** | **dbeng10** } **-krb** ...

Applies to

All operating systems except NetWare and Windows CE.

Description

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 server option" on page 156
- "-kr server option" on page 157
- "Kerberos connection parameter [KRB]" on page 229
- "Using Kerberos authentication" on page 93
- "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.

```
dbsrv10 -krb -n my_server_princ C:\kerberos.db
```

-m server option

Function

Delete the transaction log when a checkpoint is done.

Syntax

{ **dbsrv10** | **dbeng10** } **-m** ...

Applies to

All operating systems and servers.

Description

This option deletes 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 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 selected, 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.

> **Replicated and synchronized databases**
> 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.

-n server option

Function

Set the name of the database server.

Syntax

{ **dbsrv10** | **dbeng10** } **-n** *database-filename*…

Applies to

All operating systems and servers.

Description

By default, the database server receives the name of the first database file with the path and extension removed. For example, if the server is started on the file *samples-dir\demo.db* and no -n option is specified, the name of the server is demo.

The server name is interpreted according to the character set of the computer, as no database collation exists at startup time. Multibyte characters aren't recommended in server names.

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

Protocol	Truncation length
TCP/IP	250 bytes
Shared memory	250 bytes
SPX	32 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 EngineName (ENG) connection parameter of client application connection strings or profiles. With shared memory, there is a default database server that is used if no server name is specified, provided that at least one database server is 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:

```
dbsrv10 -n SERV sales.db -n DATA
```

For more information, see "-n database option" on page 199.

See also

- "Identifiers" [*SQL Anywhere Server - SQL Reference*]
- "EngineName connection parameter [ENG]" on page 225

-o server option

Function

Print all Server Messages window output to the console log.

Syntax

{ **dbsrv10** | **dbeng10** } **-o** *filename* …

Applies to

All operating systems and servers.

Description

Print all Server Messages window output to the specified console log, as well as to the Server Messages window. If you specify the -qi option with -o, all messages appear only in the console log file.

You can obtain the name of the console log by executing the following command:

```
SELECT PROPERTY ( 'ConsoleLogFile' )
```

See also

- "-os server option" on page 162
- "-ot server option" on page 163
- "-qi server option" on page 165

-oe server option

Function

Specify a file name to log startup errors, fatal errors, and assertions.

Syntax

{ **dbsrv10** | **dbeng10** } **-oe** *filename* ...

Applies to

All operating systems and servers.

Description

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 Window CE) or the Unix system log regardless of whether -oe is specified.

-on server option

Function

Specifies a maximum size for the console log, after which the file is renamed with the extension *.old* and a new file is started.

Syntax

{ **dbsrv10** | **dbeng10** } **-on** { *size*[**k** | **m** | **g**] } ...

Description

The *size* is the maximum file size for the console 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 console log reaches the specified size, the database server renames the output file with the extension *.old*, and starts a new one with the original name.

> **Note**
> If the *.old* console log already exists, it is overwritten. To avoid losing old console logs, use the -os option instead.

This option cannot be used with the -os option.

See also

- "-o server option" on page 161
- "-os server option" on page 162
- "-ot server option" on page 163

-os server option

Function

Specifies a maximum size for the console log, at which point the file is renamed.

Syntax

{ **dbsrv10** | **dbeng10** } **-os** { *size*[**k** | **m** | **g**] } ...

Applies to

All operating systems and servers.

Description

The *size* is the maximum file size for logging console 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 console log, it checks the current file size. If the log message will make the file size exceed the specified size, the database server renames the console log 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 console files that can be deleted to free up disk space.

This option cannot be used with the -on option.

See also

- "-o server option" on page 161
- "-on server option" on page 161
- "-ot server option" on page 163

-ot server option

Function

Truncates the console log and appends output messages to it.

Syntax

{ **dbsrv10** | **dbeng10** } **-ot** *logfile* …

Applies to

All operating systems and servers.

Description

The functionality is the same as the -o option except the console log is truncated before any messages are written to it. You can obtain the name of the console log using the following command:

```
SELECT PROPERTY ( 'ConsoleLogFile' )
```

See also

- "-o server option" on page 161
- "-os server option" on page 162

-p server option

Function

Set the maximum size of communication packets.

Syntax

{ **dbsrv10** | **dbeng10** } **-p** *integer* …

Applies to

All operating systems and servers.

Description

The default is 1460 bytes. The minimum value is 300 bytes and the maximum is 16000.

See also

- "CommBufferSize connection parameter [CBSIZE]" on page 210

-pc server option

Function

Compress all connections except for same-computer connections.

Syntax

{ **dbsrv10** } **-pc** ...

Applies to

All operating systems and network servers, except web servers.

Description

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

- "Adjusting communication compression settings to improve performance" on page 112
- "Compress connection parameter [COMP]" on page 213
- "Use the compression features" [*SQL Anywhere Server - SQL Usage*]

-pt server option

Function

Increase or decrease the size limit at which packets are compressed.

Syntax

{ **dbsrv10** } **-pt** *size* ...

Applies to

All operating systems and network servers.

Description

This parameter takes an integer value representing the minimum byte-size of packets to be compressed. Values less than 80 are not recommended. The default is 120 bytes.

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

- "Adjusting communication compression settings to improve performance" on page 112
- "CompressionThreshold connection parameter [COMPTH]" on page 214
- "Use the compression features" [*SQL Anywhere Server - SQL Usage*]

-qi server option

Function

Control whether database server tray icon and Server Messages window appear.

Syntax

{ **dbsrv10** | **dbeng10** } **-qi** …

Applies to

Windows XP/200x

Description

This option leaves no visual indication that the server is running, other than possible startup error dialogs. You can use either (or both) the -o or -oe logs to diagnose errors.

-qn server option

Function

Do not minimize the Server Messages window on startup.

Syntax

{ **dbsrv10** | **dbeng10** } **-qn** …

Applies to

Windows

Solaris and Linux (if X Windows Server is used)

Description

By default, the Server Messages window automatically minimizes once database server startup completes. When this option is specified, the Server Messages window does not minimize after the database server starts.

The Server Messages window may appear in the background if an application autostarting the database server it is not active and -qn is specified.

On Solaris and Linux, you must specify the -ux option (use X Windows Server) with the -qn option.

See also

- "-ux server option" on page 181
- "-qi server option" on page 165
- "-qp server option" on page 166
- "-qs server option" on page 166
- "-qw server option" on page 167

Example

The following command starts the database server on Linux or Solaris, displays the Server Messages window, and does not minimize the Server Messages window once the database server is started:

```
dbeng10 -ux -qn sample.db
```

-qp server option

Function

Do not display messages about performance in the Server Messages window.

Syntax

{ **dbsrv10** | **dbeng10** } **-qp** …

Applies to

All operating systems and servers.

Description

Do not display messages about performance in the 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

-qs server option

Function

Suppress startup error dialogs.

Syntax

{ **dbsrv10** | **dbeng10** } **-qs** ...

Applies to

Windows only.

Description

This option suppresses startup error dialogs. Startup errors include errors such as:

- Couldn't open/read database file: *database-file*
- A database server with that name has already started

On Windows platforms, if the server isn't being autostarted, these errors appear in a dialog and must be cleared before the server stops. These dialogs do not appear if the -qs option is used.

If there is an error loading the language DLL, no dialog 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 CE).

Usage errors are suppressed if -qs is on the command line, but not in @data expansion.

-qw server option

Function

Do not display database Server Messages window.

Syntax

{ **dbsrv10** | **dbeng10** } **-qw** ...

Applies to

All operating systems and servers except NetWare.

Description

This option suppresses the Server Messages window (Windows platforms) and messages on the console (non-Windows platforms).

-r server option

Function

Force all databases that start on the server to be read-only. No changes to the database(s) are allowed: the server doesn't modify the database file(s) and transaction log files.

Syntax

{ **dbsrv10** | **dbeng10** } **-r** ...

Applies to

All operating systems and servers.

Description

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 server 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.

See also

- "-r database option" on page 199
- "auditing option [database]" on page 395

Example

To open two databases in read-only mode

```
dbeng10 -r database1.db database2.db
```

To open only the first of two databases in read-only mode.

```
dbeng10 database1.db -r database2.db
```

-s server option

Function

Set the user ID for syslog messages.

Syntax

{ **dbsrv10** | **dbeng10** } **-s** { **none** | **user** | **daemon** | **local*n*** } …

Applies to

Unix

Description

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, autostarted 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, 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:

   ```
   localn.err;localn.info;localn.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:

   ```
   dbeng10 -s localn ...
   ```

   Now any messages that the SQL Anywhere database server reports to syslog are redirected to the */var/adm/sqlanywhere* file.

-sb server option

Function

Specify how the server reacts to broadcasts.

Syntax

{ **dbsrv10** | **dbeng10** } **-sb** { **0** | **1** } ...

Applies to

SPX, TCP/IP

Description

Using -sb 0 causes the server not to start up any UDP or IPX broadcast listeners. In addition to forcing clients to use the DoBroadcast=NONE and HOST= options to connect to the server, this option causes the server to be unlisted when using dblocate.

Using -sb 1 causes the server to not respond to broadcasts from dblocate, while leaving connection logic unaffected. You can connect to the server by specifying LINKS=tcpip and ENG=*name*.

-sf server option

Function

Secure features for databases running on this database server.

Syntax

{ **dbsrv10** | **dbeng10** } **-sf** *feature-list* …

Applies to

All operating systems and servers.

Description

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 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:

```
dbeng10 -n secure_server -sf all,-dbspace
```

The following *feature-name* values are supported (values enclosed in parentheses are the short forms of feature names that can also be specified):

Feature name	Description
backup	Disables the use of the BACKUP statement, and therefore, the ability to run server-side backups. This does not restrict the ability to perform client-side backups using dbbackup. See "BACKUP statement" [*SQL Anywhere Server - SQL Reference*].
database	Disables the use of the CREATE DATABASE, ALTER DATABASE, DROP DATABASE, CREATE ENCRYPTED FILE, and CREATE DECRYPTED FILE statements.

Feature name	Description
db_delete_file (delete_file)	Disables the use of the db_delete_file DBLib function, which deletes database files. db_delete_file 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*].
dbspace	Disables the use of the CREATE DBSPACE, ALTER DBSPACE, and DROP DBSPACE statements.
directory (dir)	Disables the use of directory class proxy tables. This feature is also disabled when remote_data_access is disabled.
external_procedure (ext_proc)	Disables the use of external stored procedures. This 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. See "Calling external libraries from procedures" [*SQL Anywhere Server - SQL Usage*].
java	Disables the use of Java-related features, such as Java procedures. See "Using Java in the Database" [*SQL Anywhere Server - Programming*].
load_table (load)	Disables the use of the LOAD TABLE statement. See "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*].
log_file (log)	Disables the ability to change the log file and disables the ability to increase the maximum size of the log file. You can specify a server log file and its size when starting the database server.
remote_data_access (proxy)	Disables the use of any remote data access services, such as proxy tables.
request_log (rll)	Disables the ability to change the request log file 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, as well as 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*].
restore	Disables the use of the RESTORE DATABASE statement. See "RESTORE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].
unload	Disables the use of the UNLOAD TABLE and UNLOAD statements. See "UNLOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*] and "UNLOAD statement" [*SQL Anywhere Server - SQL Reference*].
web_service_client (web_client)	Disables the use of stored procedures defined to be a remote call to an external HTTP or SOAP web service.

Feature name	Description
xp_cmdshell (cmdshell)	Disables the use of the xp_cmdshell procedure. See "xp_cmdshell system procedure" [*SQL Anywhere Server - SQL Reference*].
xp_read_file (read_file)	Disables the use of the xp_read_file procedure. See "xp_read_file system procedure" [*SQL Anywhere Server - SQL Reference*].
xp_write_file (write_file)	Disables the use of the xp_write_file procedure. See "xp_write_file system procedure" [*SQL Anywhere Server - SQL Reference*].

The following feature sets let you disable groups of related features. The following values are supported:

Feature set	Description
all	Disables all features that can be disabled (all features in the above list).
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 cmdshell, external_procedure, and java features.
local_db	Disables all features related to database files. This set consists of the backup, restore, database, and dbspace features.
local_io	Disables all features that allow direct access to files and their contents. This set consists of the db_delete_file, xp_read_file, xp_write_file, directory, load_table, and unload features.
local_log	Disables all logging features that result in creating or writing data directly to a file on disk. This set consists of the request_log and log_file features.
local	Disables all local-related features. This set consists of the local_call, local_db, local_io, and local_log features.
none	Specifies that no features are disabled.
remote	Disables all features that allow remote access or communication with remote processes. This set consists of the web_service_client and remote_data_access features.

See also

- "-sk server option" on page 173
- "secure_feature_key [database]" on page 455
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Specifying secured features" on page 866

Examples

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.

```
dbsrv10 -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:

```
dbeng10 -n secure_server -sf all,-local_db
```

-sk server option

Function

Specify a key that can be used to enable features that are disabled for the database server.

Syntax

{ **dbsrv10** | **dbeng10** } **-sk** *key* ...

Applies to

All operating systems and servers.

Description

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 server option" on page 170
- "secure_feature_key [database]" on page 455
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Specifying secured features" on page 866

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.

```
dbsrv10 -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' )
```

-su server option

Function

Set the password for the DBA user of the utility database (utility_db), or disable connections to the utility database.

Syntax

{ **dbsrv10** | **dbeng10** } **-su** *password* …

Applies to

All operating systems and servers.

Description

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 GRANT CONNECT TO 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

- "File Hiding utility (dbfhide)" on page 608
- "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- "REVOKE statement" [*SQL Anywhere Server - SQL Reference*]
- "Connecting to the utility database" on page 272

Example

The following command disables all connections to the utility database:

```
dbeng10 -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:

```
dbsrv10 -su @util_db_pwd_hide.cfg -n my_server c:\inventory.db
```

-ti server option

Function

Disconnect inactive connections.

Syntax

{ **dbsrv10** | **dbeng10** } **-ti** *minutes* …

Applies to

All operating systems and servers.

Description

Disconnect connections that haven't submitted a request for the specified number of *minutes*. The default is 240 (4 hours). 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 terminated. The -ti option is provided to disconnect inactive connections, freeing their locks.

The -ti option is very useful when used in conjunction with dbsrv10 since most connections will be over network links (TCP or SPX).

The -ti option is useful with dbeng10 only for local TCP/IP connections. Using -ti has no effect on connections to a local server using shared memory.

Setting the value to zero disables checking of inactive connections, so that no connections are disconnected.

See also

- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

-tl server option

Function

Set the period at which to send liveness packets.

Syntax

{ **dbsrv10** | **dbeng10** } **-tl** *seconds* …

Applies to

All database servers using TCP/IP or SPX.

See also

- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

Description

A liveness packet is sent periodically across a client/server TCP/IP or SPX 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. This enables the server to 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. If liveness packets take a long time to send (depending on the network, the computer's hardware, and the CPU and network load on the computer), it is possible that liveness packets will sent after two thirds of the LivenessTimeout. A warning appears in the server console 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:

```
dbsrv10 -tl 0
```

Rather than disabling the LivenessTimeout option, consider increasing the value to 1 hour as follows:

```
dbsrv10 -tl 3600
```

-tmf server option

Function

Use for recovery from distributed transactions in unusual circumstances.

Syntax

{ **dbsrv10** | **dbeng10** } **-tmf** …

Applies to

Windows XP/200x only

Description

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 aren't recovered properly. It isn't for routine use.

-tmt server option

Function

Set a reenlistment timeout for participation in distributed transactions.

Syntax

{ **dbsrv10** | **dbeng10** } **-tmt** *milliseconds* …

Applies to

Windows XP/200x only

Description

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).

-tq server option

Function

Shut down the server at a specified time.

Syntax

{ **dbsrv10** | **dbeng10** } **-tq** { *datetime* | *time* } …

Applies to

All operating systems and servers.

Description

This is useful for setting up automatic off-line backup procedures (see “Backup and Data Recovery” on page 761). 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

- “sa_server_option system procedure” [*SQL Anywhere Server - SQL Reference*]

-u server option

Function

Open files using the operating system disk cache.

Syntax

{ **dbsrv10** | **dbeng10** } **-u** …

Applies to

Windows XP/200x, Unix

Description

Files are opened using the operating system disk cache in addition to the database cache.

While the operating system disk cache may improve performance in some cases, in general better performance is obtained without this option, using the database cache only.

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 server option

Function

Turn off use of asynchronous I/O.

Syntax

{ **dbsrv10** | **dbeng10** } **-ua** ...

Applies to

Linux

Description

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 server option

Function

Start the database server in console mode. This is the default.

Syntax

{ **dbsrv10** | **dbeng10** } **-uc**

Applies to

Unix

Description

Starts the database server in console mode. You should only specify one of -uc, -ui, or -ux. When you specify -uc, this starts the database server in the same manner as previous releases of the software.

☞ For information about starting the database server as a daemon, see "-ud server option" on page 179.

See also

- "-ui server option" on page 180
- "-ux server option" on page 181

-ud server option

Function

Run as a daemon.

Syntax

{ **dbsrv10** | **dbeng10** } **-ud** …

Applies to

Unix

Description

Using this option lets you run the server so that it continues running after the current operating system session ends.

When you start the daemon directly using the -ud option, the dbeng10 and dbsrv10 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 terminate 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.

See also

- "The Start Server in Background utility" on page 680
- "Software component exit codes" [*SQL Anywhere Server - Programming*]

-uf server option

Function

Specify the action to take when a fatal error occurs.

Syntax

{ **dbsrv10** | **dbeng10** } **-uf** *action* …

Applies to

Unix

Description

Use this option to specify which of the following actions is taken when a fatal error occurs:

- **abort** the Unix abort function is called, and a core file is generated.
- **default** the database server behaves in the same manner as abort in all cases, except when a device-full fatal error occurs. In this case, it behaves in the same manner as defunct. This 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.

See also

- "-ux server option" on page 181

-ui server option

Function

Open the Server Startup Options dialog and display the Server Messages window, or start the database server in console mode if a usable display isn't available on Linux and Solaris (start the database server whether or not the X Windows Server starts).

Syntax

{ **dbsrv10** | **dbeng10** } **-ui**

Applies to

Linux with X Windows Server support

Description

The -ui option allows you to use the Server Startup Options dialog to specify server options when starting the database server and display the Server Messages window once the server has started.

When -ui is specified, the server attempts to find a usable display. If it cannot find one, for example because the DISPLAY environment variable isn't set or because X Windows Server isn't running, then the database server starts in console 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, or -ux.

When the -ux option is the only option specified on the server command line, the Server Startup Options dialog appears where you can enter options for starting the database server. The Server Messages dialog appears once the server starts. If you specify other server options in addition to -ux, then the Server Messages window appears once the database server starts.

☞ For information about starting the database server as a daemon, see "-ud server option" on page 179.

See also

- "-uc server option" on page 178
- "-ux server option" on page 181

-ut server option

Function

Touch temporary files.

Syntax

{ **dbsrv10** | **dbeng10** } **-ut** *minutes* …

Applies to

Unix

Description

This option causes the server to touch temporary files at specified intervals.

-ux server option

Function

Open the Server Startup Options dialog or display the Server Messages window on Linux and Solaris (use the X Windows Server).

Syntax

{ **dbsrv10** | **dbeng10** } **-ux**

Applies to

Linux with X Windows Server support

Description

The -ux option allows you to do two things when starting the database server: use the Server Startup Options dialog to specify server options when starting the database server and display the 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 dialog 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 Windows 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 Server Messages window appears once the database server is started. You should only specify one of -uc, -ui, or -ux.

☞ For information about starting the database server as a daemon, see “-ud server option” on page 179.

See also

♦ “-uc server option” on page 178
♦ “-ui server option” on page 180
♦ “-qn server option” on page 165

Example

The following command displays the Server Startup Options dialog where you can enter options for starting the database server:

```
dbeng10 -ux
```

The following command starts the database server and displays the Server Messages window:

```
dbeng10 -ux sample.db
```

-v server option

Function

Display the software version.

Syntax

{ **dbsrv10** | **dbeng10** } **-v** …

Applies to

All operating systems and servers.

Description

Supplies the database server version in a message box, and then stops.

-x server option

Function

Specify server side network communications protocols.

Syntax 1

dbsrv10 -x { **all** | **none** | *srv-protocols* } …

srv-protocols:
{ [**spx** | **tcpip**] *parmlist* },…
parmlist:
(*parm=value*;…**)**

Syntax 2

dbeng10 -x { **all** | **none** | *eng-protocols* } …

eng-protocols:
{ **tcpip** [*parmlist*] },…
parmlist:
(*parm*=*value*;…**)**

Applies to

All operating systems and servers.

Description

Use the -x option to specify which communications protocols, in addition to shared memory, you want to use to listen for client connection broadcasts.

If you do not specify the -x option, the server attempts to listen for client connection broadcasts using all protocols supported by the database server running on your operating system, including shared memory.

If you specify the -x option with one or more protocols, the server attempts to listen for client connection broadcasts using the specified protocol(s) and also using a shared memory protocol.

> If you are running Windows CE and specify the -x option, the server only attempts to listen for client connection broadcasts using the TCP/IP protocol unless you explicitly request otherwise.

Regardless of which settings you choose for the -x option, the server always listens for connection broadcasts using the shared memory protocol. In addition to the shared memory protocol, you can also specify the following:

- **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.
- **SPX** Listen for connection attempts by the client using the SPX protocol. The SPX protocol is supported by NetWare and Windows network servers.
- **TCPIP (TCP)** Attempt to connect to 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 ensures a connection in most cases.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). Additionally, if the client and server are communicating through a firewall, the client must send the packet to the exact port the server is listening on by specifying DoBroadcast=None and Host=.

☞ For information, see “ServerPort protocol option [PORT]” on page 258.

For some protocols, additional parameters may be provided, in the format

```
-x tcpip(PARM1=value1;PARM2=value2;...)
```

☞ For a description of available parameters, see “Network protocol options” on page 242.

For Unix, quotation marks are required if more than one parameter is supplied:

```
-x "tcpip(PARM1=value1;PARM2=value2;...)"
```

See also

- "CommLinks connection parameter [LINKS]" on page 211

Example

Allow only shared memory and TCP/IP communications:

```
-x tcpip
```

-xa server option

Function

Specify a comma-separated list of database names and authentication strings for an arbiter server.

Syntax

-xa auth=*auth-strings*;**DBN=***database-names*

Applies to

All operating systems, network server only.

Description

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

- "DatabaseName connection parameter [DBN]" on page 218
- "-sn database option" on page 200
- "-xf server option" on page 184
- "-xp database option" on page 202

Example

The following command starts an arbiter database server named arbiter.

```
dbsrv10 -x tcpip -n arbiter -xa AUTH=abc;DBN=demo -xf c:\arbiterstate.txt
```

-xf server option

Function

Specify the location of the file used for maintaining state information about your database mirroring system.

Syntax

-xf *state-file* …

Applies to

All operating systems, network server only.

Description

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. By default, the state information file is named *server-name.mirror_state*.

☞ For more information about the database mirroring state information file, see “State information files” on page 833.

See also

- “-sn database option” on page 200
- “-xa server option” on page 184
- “-xp database option” on page 202

Example

The following command (entered all on one line) starts a database server named server1, that uses the state information file *c:\server1state.txt*.

```
dbsrv10.exe -n server1 -x tcpip{DOBROADCAST=no}
-xf c:\server1state.txt mydemo.db -sn mirrordemo
-xp partner=(ENG=server2;
AUTH=abc;ARBITER=(ENG=arbiter;LINKS=tcpip(TIMEOUT=1));
MODE=sync;LINKS=tcpip(TIMEOUT=1))
```

-xs server option

Function

Specify server-side web services communications protocols.

Syntax

{ **dbeng10** | **dbsrv10** } **-xs** { **none** | *web-protocols* } …

web-protocols : { **http** [**(** *parm=value*;…**)**] | **https** ([**FIPS=**{ **ON** | **OFF** };]*parmlist*) } , ...

Applies to

All operating systems and servers.

Description

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 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.

☞ For more information, see "Using transport-layer security for SQL Anywhere web services" on page 902.

Separately licensed component required
ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.
See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].

Regardless of which settings you specify with the -xs option, the server always listens for connection attempts using the shared memory protocol. You can specify any of the following:

- **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.

  Specify **FIPS=ON** to use FIPS-approved algorithms for encryption. FIPS-approved HTTPS uses a separate approved library but is compatible with HTTPS.
- **NONE** Do not listen for web requests. This is the default.

☞ For a description of available parameters, see "Network protocol options" on page 242.

On Unix, quotation marks are required if more than one parameter is supplied:

```
-xs "HTTP(PARM1=value1;PARM2=value2;...)"
```

Example

Listen for HTTP web requests on port 80:

```
dbeng10 web.db -xs HTTP(PORT=80)
```

Listen for web requests using HTTPS:

```
dbeng10 web.db -xs HTTPS
(FIPS=OFF;PORT=82;CERTIFICATE=sample.crt;CERTIFICATE_PASSWORD=tJ1#m6+W)
```

-z server option

Function

Display diagnostic communication messages, and other messages, for troubleshooting purposes.

Syntax

{ **dbsrv10** | **dbeng10** } **-z** …

Applies to

All operating systems and servers.

Description

This should only be used when tracking problems. The information appears in the Server Messages window.

-ze server option

Function

Display database server environment variables in the Server Messages window.

Syntax

{ **dbsrv10** | **dbeng10** } **-ze** …

Applies to

All operating systems and servers except NetWare and Windows CE.

Description

When you specify the -ze option, environment variables are listed in the Server Messages window on startup. You can log the contents of the Server Messages window to a file by specifying the -o option when starting the database server.

See also

- ♦ "SQL Anywhere Environment Variables" on page 277
- ♦ "-o server option" on page 161

Example

The following command starts a database server named myserver, and outputs the environment variables set for the server to the Server Messages window and the file *server-log.txt*.

```
dbeng10 -n myserver -ze -o server-log.txt
```

-zl server option

Function

Turn on capturing of the most recently-prepared SQL statement for each connection to databases on the server.

Syntax

{ **dbsrv10** | **dbeng10** } **-zl** ...

Applies to

All operating systems and servers.

Description

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.

> **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

- "Connection-level properties" on page 476
- "sa_conn_activity system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

-zn server option

Function

Specifies the number of request log file copies to retain.

Syntax

{ **dbsrv10** | **dbeng10** } **-zn** *integer*

Applies to

All operating systems and servers.

Description

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.

☞ For more information about the -zs option, see "-zs server option" on page 192.

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

- "-zo server option" on page 189
- "-zr server option" on page 190
- "-zs server option" on page 192
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

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:

```
dbeng10 "c:\my data\mydatabase.db" -zr all -zn 3
 -zs 10 -zo mydatabase.log
```

-zo server option

Function

Redirect request logging information to a file separate from the regular log file.

Syntax

{ **dbsrv10** | **dbeng10** } **-zo** *filename*…

Applies to

All operating systems and servers.

Description

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 Server Messages window.

See also

- "-zn server option" on page 188
- "-zr server option" on page 190
- "-zs server option" on page 192

-zp server option

Function

Turn on capturing of the plan most recently used by the query optimizer.

Syntax

{ **dbsrv10** | **dbeng10** } **-zp** ...

Applies to

All operating systems and servers.

Description

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

- "Connection-level properties" on page 476
- "sa_conn_activity system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

-zr server option

Function

Enable request logging of operations.

Syntax

{ **dbsrv10** | **dbeng10** } **-zr** { **all** | **SQL** | **none** | **SQL+hostvars** } ...

Applies to

All operating systems and servers.

Description

This should only be used when tracking problems. The information appears in the Server Messages window or is sent to the logging file.

The values for -zr return the following types of information:

all This value logs SQL statements, host variables, statements executed within stored procedure, statements executed from within triggers, additional requests such as FETCH and PREFETCH, the short form of query plans, and information about when a connection is blocked and unblocked on another connection. This setting can cause the log file to grow rapidly and could negatively impact server performance.

SQL enables logging of the following:

- START DATABASE and STOP DATABASE statements
- STOP ENGINE statement
- Statement preparation and execution
- EXECUTE IMMEDIATE statements
- Option settings
- COMMIT statements
- ROLLBACK statements
- PREPARE TO COMMIT operations
- Connections and disconnections
- Beginnings of transactions
- DROP STATEMENT statement
- Cursor explanations
- Cursor open, close, and resume
- Errors

none turns off request logging.

SQL+hostvars enables logging of host variable values, as well as the information listed for the SQL parameter.

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.

☞ For more information, 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

- “-zn server option” on page 188
- “-zo server option” on page 189

-zs server option

Function

Limit the size of the request logging file.

Syntax

{ **dbsrv10** | **dbeng10** } **-zs** { *size*[**k** | **m** | **g** } ...

Applies to

All operating systems and servers.

Description

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 logging file, 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

- "-zn server option" on page 188
- "-zo server option" on page 189
- "-zr server option" on page 190
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]

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:

```
dbeng10 -zr all -zs 10 -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 server option

Function

Turn on logging of request timing information.

Syntax

{ **dbsrv10** | **dbeng10** } **-zt** …

Applies to

All operating systems and servers.

Description

Once the database server is started, you can change the status for logging of request timing information using the sa_server_option system procedure.

☞ For more information, see "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].

You can find the current value of the RequestTiming setting using the following query:

```
SELECT PROPERTY( 'RequestTiming' )
```

See also

- "sa_performance_diagnostics system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_performance_statistics system procedure" [*SQL Anywhere Server - SQL Reference*]

Recovery options

These options are for use in recovery situations only.

-f recovery option

Function

Force the database server to start after the transaction log has been lost.

Syntax

{ **dbsrv10** | **dbeng10** } **-f** …

Applies to

All operating systems and servers.

Description

If there is no transaction log, the database server performs a checkpoint recovery of the database and then terminates—it doesn't continue to run. You can then restart the database server without the -f option for normal operation.

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 terminates—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.

☞ For more information, see "Backup and Data Recovery" on page 761.

Example

The following command forces the database server to start and perform a recovery of the database *mydatabase.db*:

```
dbeng10 mydatabase.db -f
```

Database options

These options are entered after the database file, and apply only to that database.

-a database option

Function

Apply the named transaction log.

Syntax

{ **dbsrv10** | **dbeng10** } [*server-options*] *database-file* **-a** *log-filename* …

Applies to

All operating systems and servers.

Description

This 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 instead of -a.

Specifying a cache size when starting the server can reduce recovery time.

☞ For information on recovery, see "Backup and Data Recovery" on page 761.

See also

- ♦ "Recovering from media failure on the database file" on page 798
- ♦ "Recovering from multiple transaction logs" on page 803
- ♦ "-ad database option" on page 195
- ♦ "-ar database option" on page 195

Example

The following example, entered all on one line, applies the log file *demo.log* to a backup copy of the sample database.

```
dbeng10 "c:\backup\demo.db" -a "c:\backup\demo.log"
```

-ad database option

Function

Specifies the directory containing log files to be applied to the database.

Syntax

{ **dbsrv10** | **dbeng10** } [*server-options*] *database-file* **-ad** *log-directory* …

Applies to

All operating systems and servers.

Description

When you include the -ad option, the specified directory is scanned for log files associated with the database. 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 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 log files have been applied.

See also

♦ "Recovering from media failure on the database file" on page 798
♦ "Recovering from multiple transaction logs" on page 803
♦ "-a database option" on page 194
♦ "-ar database option" on page 195
♦ "-as database option" on page 196

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.

```
dbeng10 "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.

```
dbeng10 "c:\mysample.db" -ad "c:\backup" -as
```

-ar database option

Function

Specifies that any log files located in the same directory as the transaction log should be applied to the database.

Syntax

{ **dbsrv10** | **dbeng10** } [*server-options*] *database-file* **-ar** …

Applies to

All operating systems and servers.

Description

When you include the -ar option, the database server looks for 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. 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 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 log files have been applied.

See also

- "Recovering from media failure on the database file" on page 798
- "Recovering from multiple transaction logs" on page 803
- "-a database option" on page 194
- "-ad database option" on page 195
- "-as database option" on page 196

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 log files have been applied.

```
dbeng10 "c:\mysample.db" -ar -as
```

-as database option

Function

Specifies that the database should continue to run after transaction logs have been applied (used in conjunction with -ad or -ar).

Syntax

{ **dbsrv10** | **dbeng10** } [*server-options*] *database-file* { **-ad** *log-dir* | **-ar** } **-as** …

Applies to

All operating systems and servers.

Description

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.

See also

- "Recovering from media failure on the database file" on page 798
- "Recovering from multiple transaction logs" on page 803
- "-a database option" on page 194
- "-ad database option" on page 195
- "-ar database option" on page 195

Examples

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.

```
dbeng10 "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.

```
dbeng10 "c:\mysample.db" -ad "c:\backup" -as
```

-dh database option

Function

Do not display this database when the Server Enumeration utility (dblocate) is used against this server. This option is position dependent.

Syntax

{ **dbsrv10** | **dbeng10** } [*server-options*] *database-file* **-dh** …

Applies to

All platforms.

Description

The -dh database option must be provided after a database file name on the command line. 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.

☞ For more information, see "Server Enumeration utility (dblocate)" on page 648.

-ek database option

Function

Specify the key for a strongly encrypted database. This option is position dependent.

Syntax

{ **dbsrv10** | **dbeng10** } [*server-options*] *database-file* **-ek** *key* …

Description

The -ek database option must be provided after a database file name on the command line. 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 dialog so it cannot be seen in clear text, use the -ep server option. See "-ep server option" on page 142.

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 885.

See also

- “-ep server option” on page 142
- “DatabaseKey connection parameter [DBKEY]” on page 217

Example

The following example starts a database and specifies the encryption key on the command line.

```
dbsrv10 -x tcpip mydata.db -ek "Akmm9u70y"
```

-m database option

Function

Truncate the transaction log when a checkpoint is done. This option is position dependent.

Syntax

{ **dbsrv10** | **dbeng10** } [*server-options*] *database-file* **-m** …

Applies to

All operating systems and servers.

Description

Truncate (delete) the transaction log when a checkpoint is done, either at shutdown or as a result of a checkpoint scheduled by the server. This 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 (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.

Replicated databases
Do not use the -m option with databases that are being replicated. Replication inherently relies on transaction log information.

Example

The following example starts a database server named silver and loads the database *salesdata.db*. When a checkpoint is done, the transaction log is truncated.

```
dbsrv10 -n silver "c:\inventory details\salesdata.db" -m
```

-n database option

Function

Set the name of the database. This option is position dependent.

Syntax

{ **dbsrv10** | **dbeng10** } [*server-options*] *database-file* **-n** *string* …

Applies to

All operating systems and servers.

Description

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.

By default, the database receives the name of the database file with the path and extension removed. For example, if the database is started on *samples-dir\demo.db* and no -n option is specified, the name of the database is demo.

Database names cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons

Example

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.

```
dbsrv10 -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:

```
dbsrv10 -n SERV c:\mydata.db -n DATA
```

For more information, see “-n server option” on page 159.

-r database option

Function

Start the named database as read-only. No changes to the database(s) are allowed: the server doesn't modify the database file(s) and transaction log files. This option is position dependent.

Syntax

{ **dbsrv10** | **dbeng10** } [*server-options*] *database-file* **-r** …

Applies to

All operating systems and servers.

Description

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. If the -r server 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.

You cannot start a database in read-only mode if auditing is turned on.

See also

- "-r server option" on page 167
- "auditing option [database]" on page 395

Example

To open two databases in read-only mode

```
dbeng10 -r database1.db database2.db
```

To open only the first of two databases in read-only mode.

```
dbeng10 database1.db -r database2.db
```

-sn database option

Function

Provide an alternate server name for a single database running on a database server. This option is position dependent.

Syntax

{ **dbsrv10** | **dbeng10** } [*server-options*] *database-file* **-sn** *alternate-server-name*

Applies to

All operating systems, network server only.

Description

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 using the alternate server name to connect can *only* connect to the database that specified the alternate server name.

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 any other database running on that 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.

See also

- “-xa server option” on page 184
- “-xf server option” on page 184
- “-xp database option” on page 202
- “START DATABASE statement” [*SQL Anywhere Server - SQL Reference*]
- “Understanding database mirroring” on page 828

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*.

```
dbsrv10 -n myserver satest.db sample.db -sn mysample
```

You can connect to *sample.db* using any of the following connection parameters:

- ENG=myserver;DBN=sample
- ENG=mysample
- ENG=mysample;DBN=sample

You cannot connect to *satest.db* using ENG=mysample.

-xp database option

Function

Provide information to an operational server that allows it to connect to its partner and to the arbiter when database mirroring is being used. This option is position dependent.

Syntax

{ **dbsrv10** | **dbeng10** } [*server-options*] *database-file*
-xp partner=(*partner-conn*);
auth=*auth-str*;
[;**arbiter=**(*arbiter-conn*)]
[;**mode=**[**sync** | **async** | **page**]
[;**autofailover=**[**YES** | **NO**]]
[;**pagetimeout=***n*] …

Applies to

All operating systems, except Windows CE, network server only.

Description

When you specify -xp, you must also specify the location of the database mirroring state information file with the -xf option.

partner-conn 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 Specifies the authentication string used by the arbiter.

arbiter-conn 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 Specifies the synchronization mode used for database mirroring: synchronous (sync), asynchronous (async), or asyncfullpage (page).

autofailover 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 asynfullpage 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 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.

☞ For more information, see “Choosing a database mirroring mode” on page 831.

See also

- “-sn database option” on page 200
- “-xa server option” on page 184
- “-xf server option” on page 184

Example

The following command specifies parameters for the partner server named server2 and the arbiter server named arbsrv.

```
dbsrv10 -n server1 mydata.db -sn my data
-xp partner=(ENG=server2;LINKS=tcpip(TIMEOUT=1));
AUTH=abc;arbiter=(ENG=arbsrv;LINKS=tcpip(TIMEOUT=1))
```

CHAPTER 6

Connection Parameters and Network Protocol Options

Contents

About this chapter

This chapter provides a reference for the parameters that establish and describe connections from client applications to a database.

Connection parameters

This section describes each connection parameter. 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 77 and "Connection parameters passed as connection strings" on page 52.
- In an ODBC data source. See "Working with ODBC data sources" on page 64.
- In the SQL Anywhere Connect dialog. See "Connecting from SQL Anywhere utilities" on page 62.

The ODBC configuration dialog and the SQL Anywhere Connect dialog for Windows operating systems share a common format. Some of the parameters correspond to checkboxes and fields in these dialogs, while others can be entered in the text box on the Advanced tab.

Notes

- Connection parameters are case insensitive.
- 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.
- Connection parameters are case insensitive, although their values may not be (for example, file names on Unix).
- 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. For example, suppose a personal server with the name demo10 is running the sample database (which can be started with the command `dbeng10` *samples-dir*`\demo.db`). The following string returns the message `Ping database successful` if a database server named demo10 is running on the local computer and has a database named demo running:

  ```
  dbping -d -c "ENG=demo10;DBN=demo;UID=DBA;PWD=sql"
  ```

  The following command, however, 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 "ENG=other-server;UID=DBA;PWD=sql"
  ```

For more information, see "The Ping utility" on page 643.

See also

- "Notes about connection parameters" on page 73

AppInfo connection parameter [APP]

Function

To assist administrators in identifying the origin of particular client connections from a database server.

Usage

Anywhere

Values

String

Default

Empty string

Description

This connection parameter is sent to the database server from embedded SQL, ODBC, OLE DB, or ADO.NET clients and from applications using the iAnywhere JDBC driver. It is not available from 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' )
```

Clients can also specify their own string, which 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.
- **EXE** The name of the client executable (Windows and NetWare only).
- **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 2000, NetWare 5.1).
- **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 10.0.0.2023).

If you specify a debug log file in your client connection parameters, the APPINFO string is added to the file.

See also

- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73
- "request_timeout option [database]" on page 452

Examples

Connect to the sample database from Interactive SQL (the iAnywhere 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;OS='Windows 2000 Build 2195 Service Pack
3';PID=0x724;THREAD=0x6bc;EXE=c:\Program Files\SQL Anywhere 10\win32
\dbisqlg.exe;VERSION=10.0.0.2023;API=iAnywhereJDBC;TIMEZONEADJUSTMENT=-300
```

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;OS='Windows 2000 Build 2195 Service Pack
3';PID=0x8d0;THREAD=0xd74;EXE=c:\Program Files\SQL Anywhere 10\win32
\dbisqlg.exe;VERSION=10.0.0.2023;API=iAnywhereJDBC;TIMEZONEADJUSTMENT=-300;AP
PINFO=ISQL connection
```

AutoStart connection parameter [ASTART]

Function

To control whether a local database server is started if no connection is found.

Usage

Anywhere

Values

YES, **NO**

Default

YES

Description

By default, if no server is found during a connection attempt, and a database file, database name, or the START connection parameter is specified, then a database server is started on the same computer. You can turn this behavior off by setting the AutoStart (ASTART) connection parameter to NO in the connection string. The database server is not autostarted if the CommLinks [LINKS] parameter includes TCPIP or SPX.

See also

- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73
- "CommLinks connection parameter [LINKS]" on page 211

AutoStop connection parameter [ASTOP]

Function

To control whether a database is stopped as soon as there are no more open connections.

Usage

Embedded databases

Values

YES, **NO**

Default

YES

Description

By default, any server that is started from a connection string is stopped when there are no more connections to it. Also, any database that is loaded from a connection string is unloaded as soon as there are no more 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 connections to it. As a consequence, the database server remains operational as well.

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.

See also

- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73

CharSet connection parameter [CS]

Function

To specify the character set to be used on this connection.

Usage

Anywhere

Values

String

Default

The local character set.

☞ For information on how this is determined, see "Determining locale information" on page 323.

Description

If you supply a value for CharSet, the specified character set is used for the current connection. Setting CharSet=none disables character set conversion for the connection.

When unloading data, you can specify the character set using the CharSet connection parameter.

☞ For a list of valid character set values, see "Recommended character sets and collations" on page 330.

See also

- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73

CommBufferSize connection parameter [CBSIZE]

Function

To set the maximum size of communication packets, in bytes.

Usage

Anywhere

Values

Integer [**k**]

Default

If no CommBufferSize value is set, the CommBufferSize is controlled by the setting on the server, which defaults to 1460 bytes.

Description

The CommBufferSize (CBSIZE) connection parameter specifies the size of communication packets, in bytes. Use **k** to specify units of kilobytes. The minimum value of CommBufferSize is 300 bytes, and the maximum is 16000 bytes.

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 largest buffers are broken up by the network software. You should set the buffer size to be somewhat smaller than that allowed by your network because the network software may add information to each buffer before sending it over the network. The default of 1460 allows an ethernet packet to be completely filled when using TCP/IP.

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

- “How connection parameters work” on page 52
- “Connection parameter tips” on page 73

Example

To set the buffer size to 400 bytes:

```
...
CommBufferSize=400
...
```

Alternatively, you can set this parameter by entering its value in the Buffer Size text box on the Network tab of the ODBC Configuration dialog.

CommLinks connection parameter [LINKS]

Function

To specify client side network protocol options.

Usage

Anywhere. The CommLinks (LINKS) connection parameter is optional for connections to a personal server, and required for connections to a network server.

Values

String

Default

Use only the shared memory communication protocol to connect.

Description

If you do not specify a CommLinks (LINKS) connection parameter, the client searches for a 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 server running on the same computer, as is typical for applications connecting to a personal database server.

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 one or more protocols in the CommLinks (LINKS) connection parameter, the client uses the named communication protocol(s), *in the order specified*, to search for a network database server. Note that if shared memory is specified, an attempt to connect using shared memory is made first, and then the remaining communication protocols are tried in the order in which they are specified. A connection error appears and the connection attempt aborts if the connection fails to connect using a specified protocol, even if there are protocols remaining in the list to try.

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 all remaining and available communication protocols. Use this setting if you are unsure of which communication protocol (s) to use.
- **TCPIP (TCP)** Start the TCP/IP communication protocol. TCP/IP is supported on all operating systems.
- **SPX** Start the SPX communication protocol. The SPX protocol is supported for Windows and NetWare clients.

Each of these values can have additional network protocol options supplied.

☞ For a list of parameters, see “Network protocol options” on page 242.

You may want to use a specific protocol, as opposed to ALL, for the following reasons:

- The network library starts slightly faster if the client uses only necessary network protocols.
- Connecting to the database may be faster.
- You must specify the protocol explicitly if you want to tune the broadcast behavior of a particular protocol by providing additional network protocol options.

The CommLinks (LINKS) connection parameter corresponds to the database server –x option.

See also

- “Network protocol options” on page 242
- “Client/Server Communications” on page 103

- ♦ “-x server option” on page 182
- ♦ “How connection parameters work” on page 52
- ♦ “Connection parameter tips” on page 73

Examples

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 link and searches for the server on the local network. If that fails, it starts the SPX link and searches for the server over SPX.

```
CommLinks=tcpip,shmem,spx
```

The following connection string fragment starts the SPX port and searches for the server over SPX. If the search fails, it then starts the TCP link and searches for the server on the local network, as well as the host kangaroo. Note that if the server is found over SPX, the TCP link is *not* started.

```
CommLinks=spx,tcpip(HOST=kangaroo)
```

Compress connection parameter [COMP]

Function

To turn compression on or off for a connection. Compressing a connection may improve performance under some circumstances.

Usage

Anywhere except with TDS connections. TDS connections (including jConnect) do not support SQL Anywhere communication compression.

Values

YES, **NO**

In the case of a difference between client and server settings, the client setting applies.

Default

NO

If a value is not set for the Compress connection parameter, the compression status is controlled by the setting on the server, which defaults to no compression.

Description

The packets sent between a SQL Anywhere client and server can be compressed using the Compress (COMP) connection parameter. Large data transfers with highly compressible data tend to get the best compression rates.

Specify YES or NO to turn communication compression on or off for the connection. The are case insensitive.

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 server, use the -pc server option.

Note that same-computer connections over any communication link will not enable compression, even if the -pc option or COMPRESS=YES parameter is used.

See also

- "-pc server option" on page 164
- "Adjusting communication compression settings to improve performance" on page 112
- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73

Examples

The following connection string fragment turns packet compression ON:

```
Compress=YES
```

The following connection string fragment turns packet compression OFF:

```
Compress=NO
```

CompressionThreshold connection parameter [COMPTH]

Function

To increase or decrease the size limit at which packets are compressed. 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.

Usage

Anywhere except TDS. Only applies to compressed connections.

Values

Integer [**k**]

If both the client and server specify different compression threshold settings, the client setting applies.

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.

Description

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.

This value represents the minimum size, in bytes, of packets to be compressed. Use **k**to specify units of kilobytes. The minimum supported value is 1 byte, and the maximum supported value is 32767 bytes. Values less than 80 bytes are not recommended.

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 or not changing the compression threshold is beneficial.

See also

- “-pt server option” on page 164
- “Adjusting communication compression settings to improve performance” on page 112
- “How connection parameters work” on page 52
- “Connection parameter tips” on page 73

Example

Connect, with a compression threshold of 100 bytes.

```
CompressionThreshold=100
```

ConnectionName connection parameter [CON]

Function

Names a connection, to make switching to it easier in multi-connection applications.

Usage

Anywhere

Values

String

Default

No connection name.

Description

An optional parameter, providing a name for the particular connection you are making. You may leave this 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

- “How connection parameters work” on page 52
- “Connection parameter tips” on page 73

- “SET CONNECTION statement [Interactive SQL] [ESQL]” [*SQL Anywhere Server - SQL Reference*]

Example

Connect, naming the connection first-con:

```
CON=first-con
```

DatabaseFile connection parameter [DBF]

Function

For use when starting a database that is not already running. The DatabaseFile connection parameter indicates which database file you want to load and connect to.

If you want to connect to an already-running database, use the DatabaseName (DBN) parameter.

Usage

Embedded databases

Values

String

Default

There is no default setting.

Description

The DatabaseFile (DBF) connection parameter is used to load and connect to a specific database file that is not already running on a database server.

- If the database you want to connect to is not already running, use the DatabaseFile (DBF) connection parameter so the database can be started.
- 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 server from a command prompt, the working directory is the directory that you are in when entering the command. If you start the 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 autostart a database using both the database file and database name. The database server is not autostarted if the CommLinks [LINKS] parameter includes TCPIP or SPX.

You can also use UNC file names and Novell NetWare Directory Services file names.

☞ For information about using UNC file names and Novell NetWare Directory Services file names, see “The SQL Anywhere database server” on page 118.

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.

See also

- ♦ “-gd server option” on page 146
- ♦ “CommLinks connection parameter [LINKS]” on page 211
- ♦ “DatabaseName connection parameter [DBN]” on page 218
- ♦ “How connection parameters work” on page 52
- ♦ “Connection parameter tips” on page 73

Examples

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 the default location of *samples-dir*, see “The samples directory” on page 295.

The following two examples assume that you have started a database file named *cities.db*, and renamed the database Kitchener as follows:

```
dbeng10 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 connection parameter [DBKEY]

Function

To start an encrypted database with a connect request.

Usage

Anywhere

Values

String

Default

None

Description

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.

The encryption key is a string, including mixed cases, numbers, letters, and special characters. Database keys cannot include leading spaces, trailing spaces, or semicolons.

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 885.

See also

- “Configuring client applications to use transport-layer security” on page 899
- “-ec server option” on page 139
- “-ek database option” on page 197
- “-ep server option” on page 142
- “How connection parameters work” on page 52
- “Connection parameter tips” on page 73
- “Encryption connection parameter [ENC]” on page 222

Example

The following fragment illustrates the use of the DatabaseKey (DBKEY) connection parameter:

```
"UID=DBA;PWD=sql;ENG=myeng;DBKEY=V3moj3952B;DBF=samples-dir\demo.db"
```

DatabaseName connection parameter [DBN]

Function

For use when connecting to a database that is already running. Identifies a loaded database to which a connection needs to be made.

If you want to connect to a database that is not already running, use the DatabaseFile (DBF) parameter.

Usage

Running local databases or network servers

Values

String

Default

There is no default setting.

Description

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.

> **Note**
> The DatabaseName (DBN) connection parameter is recommended for naming databases, rather than using the -n option with the DatabaseSwitches (DBS) connection parameter.

If the database you want to connect to is already running, you should specify the database name rather than the database file.

A connection will only occur 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 autostart a database using both the database file and database name.

See also

- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73

Example

To start a database file named *cities.db* and rename the database Kitchener, you can use the following command:

```
dbeng10 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 connection parameter [DBS]

Function

To provide database-specific options when starting a database.

Usage

Connecting to a server when the database is not loaded. This connection parameter autostarts a server with the specified database and options if a server is not already running.

Values

String

Default

No options.

Description

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.

Only *database* options can be supplied using this parameter. Server options must be supplied using the StartLine connection parameter.

☞ For information about database options, see "Database options" on page 194.

> **Note**
> The DatabaseName (DBN) connection parameter is recommended for naming databases, rather than using the -n option with the DatabaseSwitches (DBS) connection parameter.

See also

- "The SQL Anywhere database server" on page 118
- "StartLine connection parameter [START]" on page 238
- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73

Example

The following command, entered all 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 the default location of *samples-dir*, see "The samples directory" on page 295.

DataSourceName connection parameter [DSN]

Function

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.

Usage

Anywhere

Values

String

Default

There is no default data source name.

Description

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 connection parameter [FILEDSN]" on page 226
- ♦ "How connection parameters work" on page 52
- ♦ "Connection parameter tips" on page 73
- ♦ "Using ODBC data sources on Unix" on page 67

Example

The following parameter uses a data source name:

```
DSN=My Database
```

DisableMultiRowFetch connection parameter [DMRF]

Function

To turn off multi-row fetches across the network

Usage

Anywhere

Values

YES, **NO**

Default

NO

Description

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.

☞ For more information, 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.

☞ For more information, see "Prefetching rows" [*SQL Anywhere Server - Programming*].

See also

- ♦ "How connection parameters work" on page 52

- ♦ “Connection parameter tips” on page 73

Example

The following connection string fragment prevents prefetching:

```
DMRF=YES
```

EncryptedPassword connection parameter [ENP]

Function

To provide a password, stored in an encrypted fashion in a data source.

Usage

Anywhere

Values

String

Default

None

Description

Data sources are stored on disk as a file or in the registry. Storing passwords on disk may present a security problem. For this reason, when you enter a password into a data source, it can be stored in an encrypted form.

On Unix, this information is stored in the system information file (named *.odbc.ini* by default).

☞ For more information about how the system information file is located, see “Using ODBC data sources on Unix” on page 67.

If both the Password (PWD) connection parameter and the EncryptedPassword (ENP) connection parameter are specified, Password (PWD) takes precedence.

See also

- ♦ “How connection parameters work” on page 52
- ♦ “Connection parameter tips” on page 73

Encryption connection parameter [ENC]

Function

To encrypt packets sent between the client application and the server using transport-layer security or simple encryption.

Usage

For **tls**, TCP/IP only.

For **none** or **simple**, anywhere.

Values

String

Default

none

Description

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.

☞ For more information, see "Transport-Layer Security" on page 885.

Separately licensed component required
ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.
See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].

The Encryption (ENC) connection parameter accepts the following arguments:

- **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.
- **tls** Accepts communication packets that are encrypted using one of the following algorithms:
  - **tls_type=ecc** Accepts communication packets that are encrypted using elliptic-curve based Certicom encryption technology. ECC encryption is not available on all platforms. For a list of supported platforms, see the Separately licensed components table in SQL Anywhere 10.0.0 Components by Platform. To use this type of encryption, the connection must be over the TCP/IP port. To use ECC encryption, you must generate ECC certificates.

- **tls_type=rsa;fips=n** Accepts communication packets that are encrypted using RSA encryption technology. RSA encryption is not available on all platforms. For a list of supported platforms, see the Separately licensed components table in SQL Anywhere 10.0.0 Components by Platform. To use this type of encryption, the connection must be over the TCP/IP port. To use RSA encryption, you must generate RSA certificates.
- **tls_type=rsa;fips=y** Accepts communication packets that are encrypted using RSA encryption technology. FIPS-approved RSA encryption uses a separate approved library, but is compatible with servers specifying RSA TLS with SQL Anywhere 9.0.2 or later. FIPS-approved RSA encryption is not available on all platforms. For a list of supported platforms, see the Separately licensed components table in SQL Anywhere 10.0.0 Components by Platform. To use this type of encryption, the connection must be over the TCP/IP port. To use FIPS-approved RSA encryption, you must generate RSA certificates.

The client can use the following arguments to verify the field values in the server's public certificate:

- **trusted_certificates** Specify the certificate file the client uses to authenticate the server. This is the only required parameter.
- **certificate_company** Specify the value for the organization field of the certificate.
- **certificate_unit** Specify the value for the organization unit field of the certificate.
- **certificate_name** Specify the certificate's common name.

☞ For more information about verifying certificate fields for server authentication, see "Verifying certificate fields" on page 900.

☞ For more information about using digital certificates, see "Creating digital certificates" on page 890.

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.

☞ For information about using the connection_property system function, see "CONNECTION_PROPERTY function [System]" [*SQL Anywhere Server - SQL Reference*].

See also

- "Configuring client applications to use transport-layer security" on page 899
- "-ec server option" on page 139
- "-ek database option" on page 197
- "-ep server option" on page 142
- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73
- "DatabaseKey connection parameter [DBKEY]" on page 217

Examples

The following connection string fragment connects to a database server named demo with a TCP/IP link, using transport-layer security and elliptic-curve encryption:

```
"ENG=demo;LINKS=tcpip;ENCRYPTION=tls
(tls_type=ecc;trusted_certificates=sample.crt)"
```

The following connection string fragment connects to a database server named demo with a TCP/IP link, using transport-layer security and RSA encryption:

```
"ENG=demo;LINKS=tcpip;ENCRYPTION=tls
(tls_type=rsa;fips=n;trusted_certificates=rsaserver.crt)"
```

The following connection string fragment connects to a database server named demo with a TCP/IP link, using simple encryption:

```
"ENG=demo;LINKS=tcpip;ENCRYPTION=simple"
```

EngineName connection parameter [ENG]

Function

Synonym for ServerName. The name of a running database server to which you want to connect.

Usage

Network servers or personal servers.

Values

String

Default

The default local database server.

Description

EngineName is not needed if you want to connect to the default local database server.

You need to supply an EngineName if more than one local database server is running, or if you want to connect to a network server. In the Connect dialog, and in the ODBC Administrator, this is the Server Name field.

If you are autostarting a server, you can provide a server name using this parameter.

The server name is interpreted according to the character set of the client computer. Multibyte characters are not recommended in server names.

Names must be valid identifiers. Long server names are truncated to different lengths depending on the protocol.

Protocol	Truncation length
TCP/IP	250 bytes
Shared memory	250 bytes
SPX	32 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

Note
It is recommended that you include the EngineName 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.

See also

- "Identifiers" [*SQL Anywhere Server - SQL Reference*]
- "-n server option" on page 159
- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73

Example

Connect to a server named Guelph:

```
ENG=Guelph
```

FileDataSourceName connection parameter [FILEDSN]

Function

Tells the client library there is an ODBC file data source holding information about the database to which you want to connect.

Usage

Anywhere

Values

String

Default

There is no default name.

Description

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

- "DataSourceName connection parameter [DSN]" on page 220
- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73

ForceStart connection parameter [FORCE]

Function

To start a server without attempting to connect to one.

Usage

Only with the db_start_engine function.

Values

YES, **NO**

Default

NO

Description

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*]
- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73

Idle connection parameter

Function

To specify the connection's idle timeout period.

Usage

Anywhere except with TDS and Shared Memory connections. Shared Memory and TDS connections (including jConnect) ignore the SQL Anywhere Idle (IDLE) connection parameter.

Values

Integer

Default

None

Description

The Idle (IDLE) connection parameter applies only to the current connection. You can have multiple connections on the same 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 server, which defaults to 240 minutes. In case of a conflict between timeout values, the connection timeout value supercedes any server timeout value whether specified or unspecified.

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.

See also

- “-ti server option” on page 175
- “How connection parameters work” on page 52
- “Connection parameter tips” on page 73

Example

The following connection string fragment sets the timeout value for this connection to 10 minutes:

```
"ENG=myeng;LINKS=tcpip;IDLE=10"
```

Integrated connection parameter [INT]

Function

To use the integrated login facility.

Usage

Anywhere

Values

YES, **NO**

Default

NO

Description

The Integrated (INT) connection parameter has the following settings:

- **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.

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 [database]" on page 422
- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73

Example

The following data source fragment uses an integrated login:

```
INT=YES
```

Kerberos connection parameter [KRB]

Function

To use Kerberos authentication when connecting to the database server.

Usage

All platforms except Windows CE and NetWare.

Values

YES, **NO**, **SSPI**, or *GSS-API-library-file*

Default

NO

Description

The Kerberos [KRB] connection parameter has the following settings:

- **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.
- **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.

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 server option” on page 156
- “-kr server option” on page 157
- “-krb server option” on page 157
- “Using Kerberos authentication” on page 93
- “GRANT statement” [*SQL Anywhere Server - SQL Reference*]
- “Using SSPI for Kerberos logins on Windows” on page 98

Examples

```
Kerberos=YES
Kerberos=SSPI
Kerberos=c:\Program Files\MIT\Kerberos\bin\gssapi32.dll
```

Language connection parameter [LANG]

Function

Specifies the language of the connection.

Usage

Anywhere

Values

The two-letter combination representing a language. For example, specifying LANG=DE sets the default language to German.

Default

The language specified by (in order) the SALANG environment variable, the dblang utility, or the installer.

Description

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.

☞ For a list of language codes, see “Understanding the locale language” on page 314.

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

- “How connection parameters work” on page 52
- “Connection parameter tips” on page 73

LazyClose connection parameter [LCLOSE]

Function

Enabling this option causes the `CLOSE` *cursor-name* database request to be queued, and then sent to the server with the next database request. This eliminates a network request each time a cursor is closed.

Usage

Anywhere

Values

YES, **NO**

Default

NO

Description

When this parameter is enabled, cursors are not actually closed until the next database request. Any isolation level 1 cursor stability locks still apply to the cursor while the `CLOSE` *cursor-name* database request is queued.

Enabling this option can improve performance, if your:

- network exhibits poor latency
- application sends many cursor open and close requests

Note that in rare circumstances, canceling the next request after the `CLOSE` *cursor-name* database request can leave the cursor in a state where it seems to be closed on the client side, but is not actually closed on the server side. Subsequent attempts to open another cursor with the same name will fail. Using LazyClose is not recommended if your application cancels requests frequently.

See also

- “How connection parameters work” on page 52
- “Connection parameter tips” on page 73

LivenessTimeout connection parameter [LTO]

Function

To control the termination of connections when they are no longer intact.

Usage

Network server only.

All platforms except non-threaded Unix applications.

Values

Integer, in seconds

Default

None

If no LivenessTimeout value is set, the LivenessTimeout is controlled by the setting on the server, which defaults to 120 seconds.

Description

A **liveness packet** is sent periodically across a client/server TCP/IP or SPX 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 severed.

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 of the Network tab of the ODBC Configuration dialog.

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".

See also

- “How connection parameters work” on page 52
- “Connection parameter tips” on page 73

Example

The following connection string fragment sets a LivenessTimeout value of 10 minutes:

```
LTO=600
```

LogFile connection parameter [LOG]

Function

To send client error messages and debugging messages to a file.

Usage

Anywhere

Values

String

Default

No log file.

Description

If you want to save client error messages and debugging messages in a file, use the LogFile (LOG) connection parameter.

If the file name includes a path, it is relative to the current working directory of the client application.

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:

```
Mon Aug 28 2006 12:29:46
12:29:46 Attempting to connect using:
UID=DBA;PWD=********;DBF='C:\Documents and Settings\All Users\Documents\SQL
Anywhere 10\Samples\demo.db';
ENG=demo10;START='C:\Program Files\SQL Anywhere 10\win32
\dbeng10.exe';CON='Sybase Central 1';
ASTOP=YES;LOG=c:\mylog.txt
12:29:46 Attempting to connect to a running server...
12:29:46 Trying to start SharedMemory link ...

12:29:46 SharedMemory link started successfully

12:29:46 Attempting SharedMemory connection (no sasrv.ini cached address)

12:29:46 Failed to connect over SharedMemory

12:29:46 No server found, attempting to run START line...
12:29:47 Autostarted server, attempting to connect using:
UID=DBA;PWD=********;DBF='C:\Documents and Settings\All Users\Documents\SQL
Anywhere 10\Samples\demo.db';
ENG=demo10;START='C:\Program Files\SQL Anywhere 10\win32
\dbeng10.exe';CON='Sybase Central 1';ASTOP=YES
12:29:47 Attempting SharedMemory connection (no sasrv.ini cached address)

12:29:47 Connected to server over SharedMemory

12:29:47 Connected to SQL Anywhere Server version 10.0.0.2456
12:29:47 Application information:
12:29:47 IP=10.25.99.227;HOST=mymachine-XP;OS='Windows XP Build 2600 Service
Pack 2';PID=0x21c;
THREAD=0xa38;EXE='C:\Program Files\SQL Anywhere 10\Sybase Central 5.0.0\win32
\scjview.exe';
VERSION=10.0.0.2456;API=iAnywhereJDBC;TIMEZONEADJUSTMENT=-240
12:29:47 Connected to the server, attempting to connect to a running
database...
12:29:48 [    1] Connected to database successfully
12:29:53 [    1] The number of prefetch rows has been reduced to 168 due to
```

```
the prefetch buffer
12:29:53 [    1] limit. Consider using the PrefetchBuffer connection parameter.
```

See also

- ♦ "How connection parameters work" on page 52
- ♦ "Connection parameter tips" on page 73

Example

The following command line starts Interactive SQL, connecting to the sample database with a LogFile (LOG) connection parameter:

```
dbisql -c "DSN=SQL Anywhere 10 Demo;LOG=d:\logs\test.txt"
```

Password connection parameter [PWD]

Function

To provide a password for a connection.

Usage

Anywhere

Values

String

Default

No password provided.

Description

Every user of a database has a password. The password must be supplied for the user to be allowed to connect to the database. Passwords have a maximum length of 255 bytes and are case sensitive. Passwords cannot include leading spaces, trailing spaces, or semicolons.

The Password (PWD) connection parameter is not encrypted. If you are storing passwords in a data source, you should use the EncryptedPassword (ENP) connection parameter. Sybase Central and the SQL Anywhere ODBC configuration tool both use encrypted passwords.

If both the Password (PWD) connection parameter and the EncryptedPassword (ENP) connection parameter are specified, the Password (PWD) connection parameter takes precedence.

Alternatively, you can set this parameter in the Password text box in the Connect dialog and ODBC Administrator dialog.

See also

- ♦ "EncryptedPassword connection parameter [ENP]" on page 222
- ♦ "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- ♦ "Case sensitivity" [*SQL Anywhere Server - SQL Usage*]
- ♦ "How connection parameters work" on page 52

- “Connection parameter tips” on page 73

Example

The following connection string fragment supplies the user ID DBA and password sql.

```
UID=DBA;PWD=sql
```

PrefetchBuffer connection parameter [PBUF]

Function

Set the maximum amount of memory for buffering rows, in bytes.

Usage

Anywhere

Values

Integer [**k** | **m**] between the default value and 8 MB

Default

65536 (all platforms except Windows CE)

16384 (Windows CE)

Description

The PrefetchBuffer (PBUF) connection parameter controls the memory allocated on the client to store prefetched rows. The default value is in bytes, but you can use **k** or **m** to specify units of kilobytes or megabytes, respectively. In some circumstances, increasing the number of rows prefetched from the database server by the client can improve query performance. You can increase the number of rows prefetched 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.

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, and will not be supported in future versions of the software.

☞ For more information, see “PrefetchRows connection parameter [PROWS]” on page 236.

See also

- “How connection parameters work” on page 52
- “Connection parameter tips” on page 73

Examples

The following connection string fragment could be used to determine if the PrefetchBuffer memory limit is reducing the number of rows prefetched.

```
...PrefetchRows=100;LogFile=c:\client.txt
```

The following string could be used to increase the memory limit to 256 KB:

```
...PrefetchRows=100;PrefetchBuffer=256k
```

PrefetchOnOpen connection parameter

Usage

ODBC

Values

YES, **NO**

Default

NO

Description

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
- application sends many cursor open and close requests

PrefetchRows connection parameter [PROWS]

Function

Set the maximum number of rows to prefetch when querying the database.

Usage

Anywhere

Values

Integer

Default

10

Description

Increasing the number of rows prefetched from the database server by the client can improve performance on cursors that only fetch relative 0 or 1, with either single row or wide fetches. Wide fetches include embedded SQL array fetches and ODBC block fetches.

Improvements occur particularly under the following conditions:

- The application fetches many rows (several hundred or more) with very few absolute fetches.
- The application fetches rows at a high rate, and the client and server are on the same computer or connected by a fast network.
- Client/server communication is over a slow network, such as a dial-up link or wide area network.

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.

The minimum number of rows that can be prefetched is 1, and the maximum number of rows that can be prefetched is 32767.

☞ For more information, see “PrefetchBuffer connection parameter [PBUF]” on page 235.

See also

- “How connection parameters work” on page 52
- “Connection parameter tips” on page 73

Example

The following connection string fragment sets the number of prefetched rows to 100:

```
...PrefetchRows=100;...
```

RetryConnectionTimeout connection parameter [RetryConnTO]

Function

Instruct 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.

Usage

Anywhere

Values

Integer

Default

0

Description

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. 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 `LINKS=tcp;RetryConnTO=3`, then the connection attempt still takes 5 seconds.

See also

♦ "Timeout protocol option [TO]" on page 260

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 connection parameter [ENG]

This is a synonym for the EngineName (ENG) connection parameter.

☞ For more information, see "EngineName connection parameter [ENG]" on page 225.

StartLine connection parameter [START]

Function

To start a personal database server running from an application.

Usage

Embedded databases

Values

String

Default

No StartLine parameter.

Description

You should supply a StartLine (START) connection parameter only if you are connecting to a database server that is not currently running. The StartLine connection parameter is a command line to start a personal database server. The database server is not autostarted if the CommLinks [LINKS] parameter includes TCPIP or SPX.

Note
If you want to specify the database name, database file, or server, it is recommended that you use the DBN, DBF, and ENG connection parameters, rather than the StartLine connection parameter.
The following command uses the recommended syntax:

```
START=dbeng10 -c 8M;ENG=mydb;DBN=mydb;DBF=c:\sample.db
```

The following syntax is not recommended:

```
START=dbeng10 -c 8M -n mydb "c:\sample.db"
```

☞ For a detailed description of available options, see "The SQL Anywhere database server" on page 118

Note
The StartLine connection parameter is only used to start a database server if a connection cannot be made to the specified database server or the database cannot be started and connected to on a database server that is already running. For example, suppose you start a database server running a database as follows:

```
dbeng10 c:\mydb.db
```

Connect another database (without specifying a database server name using the ENG connection parameter):

```
dbisql -c "START=dbsrv10 -c 8M;DBN=seconddb;DBF=c:
\myseconddb.db;UID=DBA;PWD=sql"
```

In this case, the dbsrv10 database server is not started. Instead, the dbeng10 database server that was used to start *mydb.db* is used to start and connect to *myseconddb.db*.
However, if ENG=*server-name* had been specified, and a server named *server-name* was not already running, then the dbsrv10 database server would have started.

See also

- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73
- "CommLinks connection parameter [LINKS]" on page 211

Example

The following data source fragment starts a personal database server with a cache of 8 MB.

```
StartLine=dbeng10 -c 8M;DBF=samples-dir\demo.db
```

☞ For information about the default location of *samples-dir*, see "The samples directory" on page 295.

Unconditional connection parameter [UNC]

Function

To stop 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.

Usage

db_stop_engine and db_stop_database functions only

Values

YES, **NO**

Default

NO

Description

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*]
- "How connection parameters work" on page 52
- "Connection parameter tips" on page 73

Userid connection parameter [UID]

Function

The user ID used to log in to the database.

Usage

Anywhere

Values

String

Default

None

Description

You must always supply a user ID when connecting to a database, unless you are using an integrated or Kerberos login.

See also

- “How connection parameters work” on page 52
- “Connection parameter tips” on page 73

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:

```
dbsrv10 -x tcpip(PARM1=value1;PARM2=value2;. . .),SPX
```

From the client side, you enter the protocol options as the CommLinks (LINKS) connection parameter:

```
CommLinks=tcpip(PARM1=value1;PARM2=value2;. . .),SPX
```

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:

```
dbsrv10 -x "tcpip(PARM1=value 1;PARM2=value 2;...),SPX"
CommLinks="tcpip(PARM1=value 1;PARM2=value 2;...),SPX"
```

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 @ server option to invoke the configuration file.

TCP/IP, HTTP, HTTPS, and SPX protocol options

The options currently available for TCP/IP, HTTP, HTTPS, and SPX are as follows.

TCP/IP	HTTP and HTTPS	SPX
Broadcast [BCAST]	Certificate	BroadcastListener [BLISTENER]
BroadcastListener [BLISTENER]	Certificate_Password	DLL
ClientPort [CPORT]	DatabaseName [DBN]	DoBroadcast [DOBROAD]
DLL	LocalOnly [LOCAL]	ExtendedName [ENAME]
DoBroadcast [DOBROAD]	LogFile [LOG]	Host [IP]
Host [IP]	LogMaxSize [LSize]	RegisterBindery [REGBIN]
LocalOnly [LOCAL]	LogOptions [LOpt]	SearchBindery [BINSEARCH]
LDAP [LDAP]	LogFormat [LF]	Timeout [TO]
MyIP [ME]	MaxConnections [MaxConn]	

TCP/IP	HTTP and HTTPS	SPX
ReceiveBufferSize [RCVBUFSZ]	MaxRequestSize [MaxSize]	
SendBufferSize [SNDBUFSZ]	MyIP [ME]	
ServerPort [PORT]	ServerPort [PORT]	
TDS	Timeout [TO]	
Timeout [TO]		
VerifyServerName [VERIFY]		

Broadcast protocol option [BCAST]

Usage

TCP/IP

Values

String, in the form of an IP address

Default

Broadcasts to all addresses on the same subnet.

Description

BROADCAST specifies the IP address that should be used to send broadcast messages. 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. See "IPv6 support in SQL Anywhere" on page 105.

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 protocol option [BLISTENER]

Usage

SPX, TCP/IP (server side)

Values

YES, **NO**

Default

YES

Description

This option allows you to turn broadcast listening OFF for this port.

Using **-sb 0** is the same as specifying `BroadcastListener=NO` on both TCP/IP and SPX.

See also

- "-sb server option" on page 169

Example

Start a database server that accepts both TCP/IP and SPX connections, but require that TCP/IP connections use the Host protocol option:

```
dbsrv10 -x tcpip(BroadcastListener=NO),spx ...
```

The following is a fragment of a client connection string to connect to the database server:

```
...LINKS=tcpip;HOST=myserver;...
```

Certificate protocol option

Usage

HTTPS

Values

String

Default

There is no default certificate name.

Description

This required option specifies the name of an encryption certificate. The password for this certificate must be specified with the Certificate_Password parameter. T

Example

Start a server that requires web connections to use a particular encryption certificate.

```
dbsrv10 -xs https(Certificate=cert.file;Certificate_Password=secret) ...
```

Certificate_Password protocol option

Usage

HTTPS

Values

String

Default

There is no default certificate password.

Description

This required option specifies the password that matches the encryption certificate specified by the Certificate parameter.

Example

Start a server that requires web connections to use a particular encryption certificate.

```
dbsrv10 -xs https(Certificate=cert.file;Certificate_Password=secret) ...
```

ClientPort protocol option [CPORT]

Usage

TCP/IP (client side only)

Values

Integer

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.

Description

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 parameter unless you need to for firewall reasons.

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 protocol option [IP]" on page 249
- "DoBroadcast protocol option [DOBROAD]" on page 247
- "ServerPort protocol option [PORT]" on page 258
- "Connecting across a firewall" on page 106

Examples

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, as well as 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 protocol option [DBN]

Usage

HTTP, HTTPS

Values

AUTO, **REQUIRED**, *database-name*

Default

AUTO

Description

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 URI.

If this parameter is set to REQUIRED, the URI must specify a database name.

If this parameter is set to AUTO, the URI may specify a database name, but does not need to do so. If the URI contains no database name, the default database on the server is used to process web requests. Since the server must guess whether or not the URI contains a database name when set to AUTO, you should design your web site so as to avoid ambiguity.

If this parameter is set to the name of a database, that database is used to process all web requests. The URI must not contain a database name.

Example

The following command starts two databases, but permits only one of them to be accessed via HTTP.

```
dbsrv10 -xs http(DBN=web) samples-dir\demo.db web.db
```

DLL protocol option

Usage

TCP/IP, SPX (Windows XP/200x) client side only.

Values

String

Default

For clients on Windows XP/200x, the default is *ws2_32.dll* (Winsock 2.2).

Description

This option is used to support untested TCP/IP protocol stacks where the required networking interface functions are in DLLs that differ from the default protocol stack. The client looks for its required functionality in the named DLLs.

For clients on Windows CE, the only supported version is *wsock32.dll* (Winsock 1.1).

Example

The following command on Windows XP uses Winsock 2.2*:*

```
dbping -c "ENG=my-eng;LINKS=tcpip(DLL=ws2_32.dll)"
```

DoBroadcast protocol option [DOBROAD]

Usage

TCP/IP, SPX

Values

ALL, **NONE**, **DIRECT** (client side)

YES, **NO** (server side)

Default

ALL (client side)

YES (server side)

Description

Client usage 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. For TCP, all broadcast packets are UDP packets. For SPX, the broadcast is only performed if the server is not found in the bindery. For SPX, all broadcast packets are IPX packets.

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.

Specifying DoBroadcast=NONE causes no UDP or IPX broadcasts to be used and the server address cache (*sasrv.ini*) is ignored. A TCP/IP or SPX connection is made directly with the HOST/PORT specified, and the server name is verified. 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 server host with the HOST option.

Server usage Setting DoBroadcast=NO prevents the database server from broadcasting to find other servers with the same name when starting up. This is useful in certain rare circumstances, but it is not generally recommended.

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
```

ExtendedName protocol option [ENAME]

Usage

SPX (platforms other than Windows XP/200x).

Values

YES, **NO**

Default

NO

Description

According to the Novell standard for legal SAP names, the following characters are not allowed:

```
\ / : ; , * ? + -
```

If you start a server named demo-1, the default behavior is to strip out the hyphen (-) and try to start a server demo1. By turning on ExtendedName, the name is left untouched.

This is an SPX port parameter and is only useful when starting a server.

Caution
Users should be wary of using this option as it is contrary to the SAP standard.

Example

The following command starts a NetWare server with the name demo-1.

```
load dbsrv10.nlm -x spx(ExtendedName=YES) demo-1
```

Host protocol option [IP]

Usage

TCP/IP, SPX

Values

String

Default

No additional computers.

Description

HOST specifies additional computers outside the immediate network to be searched by the client library. On the server, the search is performed to avoid starting a server with a duplicate name.

For TCP/IP, the address can be the *hostname* IP address. You may optionally specify a PORT value as well.

When specifying an IPv6 address on a Windows platform, the interface identifier should be used. See "IPv6 support in SQL Anywhere" on page 105.

For SPX, an address of the form *a*:*b*:*c*:*d*:*e*:*f*/*g*:*h*:*i*:*j* is used, where *a*:*b*:*c*:*d*:*e*:*f* is the node number (Ethernet card address) of the server, and *g*:*h*:*i*:*j* is the network number.

The server prints addressing information to the Server Messages window during startup if the -z option is used. In addition, the application writes this information to its log file if LogFile is specified.

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. When you specify multiple ports and servers, you can associate a particular port with a specific server by specifying the port in the HOST (IP) protocol option instead of the PORT parameter.

IP and HOST are synonyms when using TCP/IP. For SPX, you must use HOST.

See also

- "ClientPort protocol option [CPORT]" on page 245

Examples

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:

```
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)
```

LDAP protocol option [LDAP]

Usage

TCP/IP

Values

YES, **NO**, or *filename*

Default

ON

The default file name is *saldap.ini*.

Description

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.

Specifying LDAP=*filename* turns LDAP support on and uses the specified file as the configuration file. Specifying LDAP=YES turns LDAP support on and uses *saldap.ini* as the configuration file.

You can hide the contents of the *saldap.ini* file with simple encryption using the File Hiding utility.

☞ For more information, see “Hiding the contents of .ini files” on page 608.

LDAP is only used with TCP/IP.

See also

- “Connecting using an LDAP server” on page 108

LocalOnly protocol option [LOCAL]

Usage

TCP/IP, HTTP, HTTPS

Values

YES, **NO**

Default

NO

Description

The LocalOnly (LOCAL) protocol option allows a client to choose to connect only to a server on the local computer, if one exists. If no server with the matching server name is found on the local computer, a server will not be autostarted.

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.

See also

- “Broadcast protocol option [BCAST]” on page 243

LogFile protocol option [LOG]

Usage

HTTP, HTTPS

Values

Filename

Default

None

Description

Specify the name of the file to which the database server is to write information about web requests.

See also

- “LogFormat protocol option [LF]” on page 252
- “LogMaxSize protocol option [LSIZE]” on page 253
- “LogOptions protocol option [LOPT]” on page 253

LogFormat protocol option [LF]

Usage

HTTP, HTTPS

Values

Format-string

Default

@T - @W - @I - @P - "@M @U @V" - @R - @L - @E

Description

This parameter controls the format of messages written to the log file and which fields appear in them. If they appear in the string, the current values are substituted for the following codes as each message is written.

- **@@** 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.
- **@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.

See also

- "LogFile protocol option [LOG]" on page 251
- "LogMaxSize protocol option [LSIZE]" on page 253
- "LogOptions protocol option [LOPT]" on page 253

LogMaxSize protocol option [LSIZE]

Usage

HTTP, HTTPS

Values

Integer [**k** | **m** | **g**]

Default

0

Description

When the log file reaches the stated size, it is renamed and another log file is created. If LogMaxSize is zero, the log file size is unlimited. The default value is in bytes, but you can use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively.

See also

- "LogFile protocol option [LOG]" on page 251
- "LogFormat protocol option [LF]" on page 252
- "LogOptions protocol option [LOPT]" on page 253

LogOptions protocol option [LOPT]

Usage

HTTP, HTTPS

Values

NONE, **OK**, **INFO**, **ERRORS**, **ALL**, *status-codes*, **REQHDRS**, **RESHDRS**, **HEADERS**

Default

ALL

Description

The values available include keywords that select particular types of messages, and HTTP status codes. Multiple values may be specified, separated by commas.

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 (30x HTTP status codes).
- **ERRORS** Log all errors (40x and 50x 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

In addition, the following keywords may 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).

See also

- "LogFile protocol option [LOG]" on page 251
- "LogFormat protocol option [LF]" on page 252
- "LogMaxSize protocol option [LSIZE]" on page 253

MaxConnections protocol option [MAXCONN]

Usage

HTTP, HTTPS

Values

Size

Default

5 (personal server)

Number of licensed connections (network server)

Description

The number of simultaneous connections accepted by the server. The value 0 indicates no limit.

See also

- "MaxRequestSize protocol option [MAXSIZE]" on page 255

MaxRequestSize protocol option [MAXSIZE]

Usage

HTTP, HTTPS

Values

Integer [**k** | **m** | **g**]

Default

100k

Description

The size of the largest request accepted by the server. The default value is in bytes, but you can use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively. If the size of a request exceeds this limit, the connection is closed and a `413 ENTITY TOO LARGE` response is returned to the client. This value limits only the size of the request, not that of the response. The value 0 disables this limit, but should be used with extreme caution. Without this limit, a rogue client could overload the server or cause it to run out of memory.

See also

- "MaxConnections protocol option [MAXCONN]" on page 254

Example

The following command line (entered all on one line) instructs the server to accept requests up to 150000 bytes in size:

```
dbsrv10 -xs http{MaxRequestSize=150000}
```

MyIP protocol option [ME]

Usage

TCP/IP, HTTP, HTTPS

Values

String

Description

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.

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 server.

Separate multiple IP addresses with commas.

When specifying an IPv6 address on a Windows platform, the interface identifier should be used. See "IPv6 support in SQL Anywhere" on page 105.

Example

The following command line (entered all on one line) instructs the server to use two network cards.

```
dbsrv10 -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:

```
dbsrv10 -x tcpip(MyIP=fe80::5445:5245:444f) "samples-dir\demo.db"
```

☞ For information about the default location of *samples-dir*, see "The samples directory" on page 295.

The following connection string fragment instructs the client to make no attempt to determine addressing information.

```
LINKS=tcpip(MyIP=NONE)
```

ReceiveBufferSize protocol option [RCVBUFSZ]

Usage

TCP/IP

Values

Integer [**k** | **m** | **g**]

Default

Computer-dependent.

Description

Sets the size for a buffer used by the TCP/IP protocol stack. You may want to increase the value if BLOB performance over the network is important. By default, the specified buffer size is in bytes. Use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively.

RegisterBindery protocol option [REGBIN]

Usage

SPX (server side only)

Values

YES, **NO**

Default

YES

Description

The database server attempts to register its name with any active binderies on the network when loading the SPX link. To disable this name registration, set RegisterBindery to NO. In this case, the client library must be able to locate the database server over SPX by broadcasting packets.

SearchBindery protocol option [BINSEARCH]

Usage

SPX

Values

YES, **NO**

Default

YES

Description

The database server normally searched the bindery for existing database servers with the same name. When you specify SEARCHBINDERY=NO, the NetWare bindery is not searched for a database server.

SendBufferSize protocol option [SNDBUFSZ]

Usage

TCP/IP

Values

Integer [**k** | **m** | **g**]

Default

Computer-dependent.

Description

Sets the size for a buffer used by the TCP/IP protocol stack. The default value is in bytes, but you can use **k**, **m**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively. You may want to increase the value if BLOB performance over the network is important.

ServerPort protocol option [PORT]

Usage

TCP/IP, HTTP, HTTPS

Values

Integer

Default

The default value for TCP/IP is 2638. The default value for HTTP is 80. The default value for HTTPS is 443.

Description

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.

In the case of the 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. For example:

- (port=1234)
- (port=1234,1235,1239)
- (port=1234-1238)
- (port=1234-1237,1239,1242)

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.

The database server always listens on UDP port 2638 on most operating systems, even if you specify a different port using a network protocol option. Hence, applications can connect to the database server without specifying a port number.

For a client, the ServerPort 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 server.

If you using a web server, by default, the database server listens on the standard HTTP and HTTPS ports of 80 and 443, respectively.

If you start a database server using TCP/IP port number 2638 (the default), then the server also listens to UDP port 2638. 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 the database server's TCP/IP port number is not 2638, then the server listens to the same UDP port as the TCP/IP port.

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 server is not using the default port (2638). For example if the database server is started with the command `dbsrv10 -n MyServer` *samples-dir*`/demo.db`, a client on the same subnet can find the server using the following connection parameters `ENG=MyServer;LINKS=tcpip`. If another server is started on Mac OS X, with the following command `dbsrv10 -n SecondServer -x tcpip(PORT=7777)` *samples-dir*`/demo.db`, a client on the same subnet can find the server using the connection parameters `ENG=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.

Example

The following example shows how to use the PORT protocol option to specify the port the server starts on.

1. Start a network database server:

   ```
   dbsrv10 -x tcpip -n server1
   ```

   Port number 2638 is now taken.

2. Attempt to start another database server:

   ```
   dbsrv10 -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:

   ```
   dbsrv10 -xs http(port=8080) -n server3 web.db
   ```

TDS protocol option

Usage

TCP/IP (server side only)

Values

YES, **NO**

Default

YES

Description

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 server, this protocol option is the only way to disallow TDS connections.

Example

The following command starts a database server using the TCP/IP protocol, but disallowing connections from Open Client or jConnect applications.

```
dbsrv10 -x tcpip(TDS=NO) ...
```

Timeout protocol option [TO]

Usage

TCP/IP, SPX, HTTP, HTTPS

Values

Integer, in seconds

Default

5 for TCP/IP.

30 for HTTP and HTTPS.

Description

Timeout specifies the length of time, in seconds, to wait for a response when establishing communications. It 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 or SPX 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 `408 REQUEST TIMEOUT` is returned to the client. 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.

Example

The following data source fragment starts a TCP/IP communication link only, with a timeout period of twenty seconds.

```
...
CommLinks=tcpip(TO=20)
...
```

VerifyServerName protocol option [VERIFY]

Usage

TCP/IP (client side only)

Values

YES, **NO**

Default

YES

Description

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's 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.

> **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 protocol option [DOBROAD]” on page 247

Part II. 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.

CHAPTER 7

Working with Database Files

Contents

About this chapter

This chapter describes how to create and work with database and associated files.

Overview of database files

Basic 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*.

  For information on creating databases, see "Working with databases" [*SQL Anywhere Server - SQL Usage*].

- **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*.

  For information on the transaction log, see "The transaction log" on page 766.

- **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 is defined, SQL Anywhere places its temporary file in the current directory on Windows operating systems, or in the */tmp* directory on Unix.

  The 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.

  For more information, see "sa_disk_free_space system procedure" [*SQL Anywhere Server - SQL Reference*].

Additional files

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

- **Dbspace files** You can spread your data over several separate files, in addition to the database file.

  For information on dbspaces, see "CREATE DBSPACE statement" [*SQL Anywhere Server - SQL Reference*].

- **Transaction log mirror files** For additional security, you can create a mirror copy of the transaction log. This file typically has the extension *.mlg*.

  For information on mirrored transaction logs, see “Protecting against media failure on the transaction log” on page 779.

Using additional dbspaces

Typically needed for large databases
For most databases, a single database file is sufficient. 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**. All database objects and all data are placed, by default, 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 (an initial 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.

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 many commonly-employed file systems restrict file size to a maximum of 2 GB, some, such as the Windows XP/200x file system 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 "Size and number limitations" on page 518.

Splitting existing databases

If you want to split existing database objects among multiple dbspaces, you need to unload your database and modify the generated command file (named *reload.sql* by default) for rebuilding the database. To do so, 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.

See also

- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "UNLOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]

Creating a dbspace

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 may be 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. You can specify the dbspace in which an object is stored by setting the default_dbspace database option.

Placing tables in dbspaces

A newly-created dbspace is empty. When you create a new table you can place it in a specific dbspace with an IN clause in the CREATE TABLE 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 appears in the main dbspace.

Each table is entirely contained 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 INDEX statement.

♦ To create a dbspace (Sybase Central)

1. Open the Dbspaces folder for that database.

2. From the File menu, choose New ► Dbspace.

   The Create Dbspace wizard appears.

3. Follow the instructions in the wizard.

   The new dbspace appears in the Dbspaces folder.

♦ To create a dbspace (SQL)

- Execute a CREATE DBSPACE statement.

Examples

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

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

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

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

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

```
CREATE DBSPACE library
AS 'e:\\dbfiles\\library.db';
SET OPTION default_dbspace = 'library';
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 [database]” on page 409
- “Creating tables” [*SQL Anywhere Server - SQL Usage*]
- “CREATE INDEX statement” [*SQL Anywhere Server - SQL Reference*]

Pre-allocating space for database files

When you create a new database file, you can pre-allocate 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 614.

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 pre-allocate disk space for dbspaces or for transaction logs using either Sybase Central or the ALTER DBSPACE statement.

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

Performance tip
Running a disk defragmentation utility after pre-allocating 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 pre-allocate space (Sybase Central)

1. Open the Dbspaces folder.
2. Right-click the desired dbspace and choose Pre-allocate Space from the popup menu.
3. 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).
4. Click OK.

♦ To pre-allocate 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 a dbspace” on page 268
- “ALTER DBSPACE statement” [*SQL Anywhere Server - SQL Reference*]

Deleting 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. Open the Dbspaces folder.
2. Right-click the desired dbspace and choose Delete from the popup menu.

♦ 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 statement” [*SQL Anywhere Server - SQL Reference*]

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.

For example, executing the following statement after having connected to the utility database creates a database named *new.db* in the directory *c:\temp*.

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

For more information on the syntax of this statement, 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 a list of connection and server properties, see "Understanding database properties" on page 476.

Allowed statements for the utility database

The following are the only statements that you can executed when connected to the utility database:

- ALTER DATABASE *dbfile* ALTER TRANSACTION LOG
- CREATE DATABASE
- CREATE DECRYPTED FILE
- CREATE ENCRYPTED FILE
- DROP DATABASE
- GRANT CONNECT TO DBA IDENTIFIED BY *new-password*
- RESTORE DATABASE
- REVOKE CONNECT FROM DBA
- START DATABASE
- STOP DATABASE
- STOP ENGINE
- SELECT (with no FROM or WHERE clauses)

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 th -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:

```
dbsrv10 -su @util_db_pwd_hide.cfg -n my_server c:\mydb.db
```

☞ For more information about using dbfhide, see "File Hiding utility (dbfhide)" on page 608.

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 server option" on page 174.

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

1. Start a database server with the following command:

   ```
   dbeng10.exe -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 dialog, enter **DBA** as the user ID, and enter any non-blank password. The password itself is not checked, but it must not be empty.

4. On the Database tab, enter **utility_db** as the database name and **TestEng** as the server name.

5. Click OK to 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:

   ```
   dbsrv10.exe -n TestEng -su 9Bx231K
   ```

2. Start Interactive SQL.

3. In the Connect dialog, enter **DBA** as the user ID, and enter the password specified by the -su option.

4. On the Database tab, enter **utility_db** as the database name and **TestEng** as the server name.
5. Click OK to connect.

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

☞ For more information, see "Connecting to a Database" on page 49 and "-su server option" on page 174.

> **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)

> **Note**
> Because the use of the *util_db.ini* file is deprecated, it is recommended that you use the -su server option to specify the DBA password for the utility database.

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 608.

Permission 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.

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

☞ For more information on the -gu database server option, see "-gu server option" on page 154.

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 or not they can load and connect to the utility database.

```
dbsrv10.exe –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 at the command prompt with the following command.

```
dbsrv10 –n TestSrv -su secret –gu utility_db
```

The following command starts the Interactive SQL utility as a client application, connects to the server named TestSrv, loads the utility database, and connects the user.

```
dbisql -c "UID=DBA;PWD=secret;DBN=utility_db;ENG=TestSrv"
```

Having executed the above command successfully, the user connects to the utility database, and can execute file administration statements.

CHAPTER 8

SQL Anywhere Environment Variables

Contents

About this chapter

This chapter describes the environment variables used by SQL Anywhere, as well as how to set them.

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 server option" on page 187.

Setting environment variables on Windows

The SQL Anywhere installer creates or modifies the following environment variables in your computer's properties: PATH, SQLANY10, and SQLANYSH10. 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 for the Finder in Mac OS X

The SQL Anywhere installer sets the following environment variables: DYLD_LIBRARY_PATH, ODBCINI, PATH, SQLANY10, and SQLANYSH10. After installing SQL Anywhere, you must log out and log in for the environment variable settings to take effect. Rebooting is not required.

You must run the dbmodenv utility to configure the GUI environment (*~/.MacOSX/environment.plist*) for each user who wants to use SQL Anywhere applications from the Finder. Each user must log out and log in for the settings to take effect.

The dbmodenv utility is located in */Applications/SQLAnywhere10/System/bin32*.

Terminal sessions do not inherit environment variables from the Finder. The following section describes how to set environment variables for terminal sessions.

Setting environment variables on Unix and Mac OS X

Once SQL Anywhere 10 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 uncomment sections, 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 Windows, 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.

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.

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/sqlanywhere10*, the following statement sources *sa_config.sh*:

```
. /opt/sqlanywhere10/bin32/sa_config.sh
```

Under C-shell and its derivatives, the command is `source`. For example, if SQL Anywhere is installed in */opt/sqlanywhere10*, the following statement sources *sa_config.csh*:

```
source /opt/sqlanywhere10/bin32/sa_config.csh
```

DYLD_LIBRARY_PATH environment variable [Mac OS X]

Syntax

DYLD_LIBRARY_PATH=*path-list*

Default

/Applications/SQLAnywhere10/System/lib32

Description

On Mac OS X, the DYLD_LIBRARY_PATH environment variable specifies the directories that are searched at run time for libraries required by SQL Anywhere applications.

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 280
- "LIBPATH environment variable [AIX]" on page 280
- "SHLIB_PATH environment variable [HP-UX]" on page 286
- "Setting environment variables on Unix and Mac OS X" on page 278

LD_LIBRARY_PATH environment variable [Linux and Solaris]

Syntax

LD_LIBRARY_PATH=*path-list*

Default

- */opt/sqlanywhere10/lib32* (32-bit platforms)
- */opt/sqlanywhere10/lib64* (64-bit platforms)

Description

On Linux and Solaris, the LD_LIBRARY_PATH environment variable specifies the directories that are searched at run time for libraries required by SQL Anywhere applications.

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 279
- "LIBPATH environment variable [AIX]" on page 280
- "SHLIB_PATH environment variable [HP-UX]" on page 286
- "Setting environment variables on Unix and Mac OS X" on page 278

LIBPATH environment variable [AIX]

Syntax

LIBPATH=*path-list*

Default

- */usr/lpp/sqlanywhere10/lib32* (32-bit platforms)
- */usr/lpp/sqlanywhere10/lib64* (64-bit platforms)

Description

On AIX, the LIBPATH environment variable specifies the directories that are searched at run time for libraries required by SQL Anywhere applications.

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 279
- "LD_LIBRARY_PATH environment variable [Linux and Solaris]" on page 280
- "SHLIB_PATH environment variable [HP-UX]" on page 286
- "Setting environment variables on Unix and Mac OS X" on page 278

ODBCHOME environment variable [Unix]

Syntax

ODBCHOME=*odbc-ini-directory*

Description

The ODBCHOME environment variable specifies the location of *.odbc.ini*, the system information file containing ODBC data sources. If the file is named anything other than *.odbc.ini*, you must use the ODBCINI or ODBC_INI environment variables to specify its location.

☞ For the algorithm for locating ODBC data sources, see "Using ODBC data sources on Unix" on page 67.

See also

- "ODBCINI and ODBC_INI environment variables [Unix]" on page 281
- "Setting environment variables on Unix and Mac OS X" on page 278

ODBCINI and ODBC_INI environment variables [Unix]

Syntax

ODBCINI=*odbc-ini-file*

ODBC_INI=*odbc-ini-file*

Description

The ODBCINI and ODBC_INI environment variables specify the path and name of the system information file containing ODBC data sources. 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 the algorithm for locating ODBC data sources, see "Using ODBC data sources on Unix" on page 67.

See also

- "ODBCHOME environment variable [Unix]" on page 281
- "Setting environment variables on Unix and Mac OS X" on page 278

PATH environment variable

Syntax

PATH=*path-list*

Default

Note: the following paths are only added if the corresponding component is installed.

Windows XP/200x:

- *C:\Program Files\SQL Anywhere 10\win32* (Windows 32-bit)
- *C:\Program Files\SQL Anywhere 10\ia64* (Windows ia64/Itanium)
- *C:\Program Files\SQL Anywhere 10\x64* (Windows x64)
- *C:\Program Files\SQL Anywhere 10\Sybase Central 5.0.0\win32*

Mac OS X:

- */Applications/SQLAnywhere10/System/bin32*
- */Applications/SQLAnywhere10/System/sybcentral500*
- */Applications/SQLAnywhere10/System/openserver/OCS-15_0*

AIX:

- */usr/lpp/sqlanywhere10/bin32* (32-bit platforms)
- */usr/lpp/sqlanywhere10/bin64* (64-bit platforms)
- *usr/lpp/sqlanywhere10/openserver/OCS-15_0/bin*

Other Unix operating systems:

- */opt/sqlanywhere10/bin32* (32-bit platforms)
- */opt/sqlanywhere10/bin64* (64-bit platforms)
- */opt/sqlanywhere10/sybcentral500*
- */opt/sqlanywhere10/openserver/OCS-15_0/bin*

Description

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 278
- "Setting environment variables on Unix and Mac OS X" on page 278

SACHARSET environment variable

Syntax

SACHARSET=*charset*

Description

The *charset* is a character set name.

☞ For a list of recommended character sets, see "Recommended character sets and collations" on page 330.

If SACHARSET is not specified, the character set comes from the operating system.

See also

- "Setting environment variables on Windows" on page 278
- "Setting environment variables on Unix and Mac OS X" on page 278

SADIAGDIR environment variable

Syntax

SADIAGDIR=*diagnostic-information-directory*

Default

Windows: *%ALLUSERSPROFILE%\Application Data\SQL Anywhere 10\diagnostics*

Unix: *$HOME/.sqlanywhere10/diagnostics*

NetWare: *SYS:\SYSTEM*

Windows CE: directory where the database server is running

Description

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.

On Windows (except Windows CE), 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 285 and "TMP, TEMPDIR, and TEMP environment variables" on page 290.

On Windows CE, 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 CE" on page 301.

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/.sqlanywhere10/diagnostics*.

3. The current directory.

4. The temporary directory. See “SATMP environment variable” on page 285 and “TMP, TEMPDIR, and TEMP environment variables” on page 290.

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

- “The SQL Anywhere Support utility” on page 672
- “Error reporting in SQL Anywhere” on page 46
- “Setting environment variables on Windows” on page 278
- “Setting environment variables on Unix and Mac OS X” on page 278

SALANG environment variable

Syntax

SALANG=*language-code*

Description

The *language-code* is a two-letter combination representing a language. For example, setting **SALANG=DE** sets the default language to German.

☞ For a list of supported language codes, see “Understanding the locale language” on page 314.

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 “The Language Selection utility” on page 628.

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 628
- “Registry settings on installation” on page 300
- “Setting environment variables on Windows” on page 278
- “Setting environment variables on Unix and Mac OS X” on page 278

SALOGDIR environment variable

Syntax

SALOGDIR=*directory-name*

Description

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 created in the first writable location in the following list:

1. The SALOGDIR environment variable.
2. The installation directory.

   On 32-bit Windows platforms, the default location is *install-dir\win32*. If this directory does not exist, an error is given.
3. The directory of the database server executable.
4. Write the *backup.syb* file in the root directory of the current drive.

On Unix, the *backup.syb* file is created in the first writable location in the following list:

1. The SALOGDIR environment variable.
2. The HOME environment variable.
3. Write the *backup.syb* file to the directory where the database server was started.

See also

- "BACKUP statement" [*SQL Anywhere Server - SQL Reference*]
- "Setting environment variables on Windows" on page 278
- "Setting environment variables on Unix and Mac OS X" on page 278

SATMP environment variable

Syntax

SATMP=*directory-name*

Description

The SATMP file specifies the location of the temporary file 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, 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 Unix, both the client and the database server must set SATMP to the same value when connecting via shared memory.

On Windows CE, 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 CE, see "Registry settings on Windows CE" on page 301.

See also

- "-dt server option" on page 138
- "TMP, TEMPDIR, and TEMP environment variables" on page 290
- "Setting environment variables on Windows" on page 278
- "Setting environment variables on Unix and Mac OS X" on page 278
- "Place different files on different devices" [*SQL Anywhere Server - SQL Usage*]

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.

SHLIB_PATH environment variable [HP-UX]

Syntax

SHLIB_PATH=*path-list*

Default

- */opt/sqlanywhere10/lib32* (32-bit platforms)
- */opt/sqlanywhere10/lib64* (64-bit platforms)

Description

On HP-UX, the SHLIB_PATH environment variable specifies the directories that are searched at run time for libraries required by SQL Anywhere applications.

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 279
- "LD_LIBRARY_PATH environment variable [Linux and Solaris]" on page 280
- "LIBPATH environment variable [AIX]" on page 280

- “Setting environment variables on Unix and Mac OS X” on page 278

SQLANY10 environment variable

Syntax

SQLANY10=*directory-name*

Default

Windows: *C:\Program Files\SQL Anywhere 10*

AIX: */usr/lpp/sqlanywhere10*

Mac OS X: */Applications/SQLAnywhere10/System*

Other Unix operating systems: */opt/sqlanywhere10*

Description

The SQLANY10 environment variable specifies the location of the directory containing SQL Anywhere 10. 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 SQLANY10 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 278
- “Setting environment variables on Unix and Mac OS X” on page 278

SQLANYSAMP10 environment variable

Syntax

SQLANYSAMP10=*directory-name*

Default

Windows XP/200x: *C:\Documents and Settings\All Users\Documents\SQL Anywhere 10\Samples*

Windows Vista: *C:\Users\Public\Documents\SQL Anywhere 10\Samples*

Mac OS X: */Applications/SQLAnywhere10/Samples*

AIX: */usr/lpp/sqlanywhere10/samples*

Other Unix operating systems: */opt/sqlanywhere10/samples*

Description

The SQLANYSAMP10 environment variable specifies the location of the samples directory.

On Windows, the installer sets the location of the SQLANYSAMP10 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 278
- "Setting environment variables on Unix and Mac OS X" on page 278

SQLANYSH10 environment variable

Syntax

SQLANYSH10=*directory-name*

Default

Windows: *C:\Program Files\SQL Anywhere 10*

Mac OS X: */Applications/SQLAnywhere10/System*

AIX: */usr/lpp/sqlanywhere10*

Other Unix operating systems: */opt/sqlanywhere10*

Description

The SQLANYSH10 environment variable specifies the location of the shared components directory. Interactive SQL, Sybase Central, and the SQL Anywhere Console utility (dbconsole) use this variable to determine the location of the shared components directory.

On Windows, the installer sets the location of the SQLANYSH10 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 278
- "Setting environment variables on Unix and Mac OS X" on page 278

SQLCONNECT environment variable

Syntax

SQLCONNECT=*parameter*=*value*; …

Description

SQLCONNECT specifies connection parameters that are used by several of the database administration utilities when connecting to a database server. This string is a list of parameter settings, of the form **parameter**=*value*, delimited by semicolons. For a description of the connection parameters, see "Connection parameters" on page 206.

> **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 278
- "Setting environment variables on Unix and Mac OS X" on page 278

SQLPATH environment variable

Syntax

SQLPATH=*path-list*

Description

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 278
- "Setting environment variables on Unix and Mac OS X" on page 278

SQLREMOTE environment variable

Syntax

SQLREMOTE=*path*

Description

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 CE, 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 278
- "Setting environment variables on Unix and Mac OS X" on page 278

SYBASE environment variable

Syntax

SYBASE=*directory-name*

Description

The SYBASE variable marks the home directory for installation of some Sybase applications, including Adaptive Server Enterprise, Open Client, Open Server, and utilities such as DSEdit. You need this variable only if you are using other Sybase applications.

See also

- ♦ "Setting environment variables on Windows" on page 278
- ♦ "Setting environment variables on Unix and Mac OS X" on page 278

TMP, TEMPDIR, and TEMP environment variables

Syntax

TMP=*path*

TMPDIR=*path*

TEMP=*path*

Description

SQL Anywhere software may create temporary files for various operations. 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 285.

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 CE, 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 CE" on page 301.

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 server option" on page 138

- "SATMP environment variable" on page 285
- "Setting environment variables on Windows" on page 278
- "Setting environment variables on Unix and Mac OS X" on page 278
- "Place different files on different devices" [*SQL Anywhere Server - SQL Usage*]

CHAPTER 9

File Locations and Installation Settings

Contents

About this chapter

This chapter describes the installation and operating system settings used by SQL Anywhere. Depending on the operating system, these settings may be stored as environment variables, initialization file entries, or registry settings.

Installation directory structure

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 utilities provided with the SQL Anywhere product, however, are installed in other directories. This section describes only the installation directory structure for SQL Anywhere itself. The SQLANY10 environment variable specifies the location of the installation directory. See "SQLANY10 environment variable" on page 287.

The 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 Novell NetWare and Windows CE, there are several directories under the installation directory:

- **Executable directories** There is a separate directory for each operating system platform, which holds executables, dynamic link libraries, and help files.

  On Windows, except Windows CE, these files are installed in the *win32*, *ia64*, or *x64*directory. If you are using Unix, they are installed in the *bin32* or *bin64*, and *lib32* or *lib64* directories.

  You will not always have all of these directories on your computer; you will have only the ones required for your operating system version.

- **java directory**
  Java classes are stored in this directory.

- **scripts directory**
  The scripts directory contains SQL scripts that are used by the database administration utilities and as examples. If the scripts directory is not present, the administration utilities will not work.

- **h directory** The *h* directory contains header files for developing C/C++ applications for SQL Anywhere. On Unix, this directory is called *include*.

Novell NetWare file locations

On Novell NetWare, all files are installed in a single directory on the server. Throughout this documentation, when reference is made to files in subdirectories of the installation directory, the file on NetWare is in the installation directory itself.

Windows CE file locations

On Windows CE, all files are installed in the installation directory *\Program Files\SQLAny10*, 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.

The samples directory

When you install SQL Anywhere 10, you can choose the directory where the samples are installed. The documentation refers to this location as *samples-dir*.

The SQLANYSAMP10 environment variable specifies the location of *samples-dir*. See "SQLANYSAMP10 environment variable" on page 287.

On Windows, you can access the samples from the Start menu by choosing Programs ► SQL Anywhere 10 ► 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 10\Samples*	*C:\Documents and Settings\All Users \Documents\SQL Anywhere 10\Samples*[1]
Windows Vista	*%PUBLIC%\Documents\SQL Anywhere 10\Samples*	*C:\Users\Public\Documents\SQL Anywhere 10\Samples*
Windows CE	*\Program Files\SQLAny10*	
Unix and Linux	*/opt/sqlanywhere10/samples*	Script provided to copy samples to user-specified directory.
Mac OS X	*/Applications/SQLAnywhere10/Samples*	Script provided to copy samples to user-specified directory.
NetWare	No samples installed.	

[1] When accessing the SQL Anywhere samples directory in Windows Explorer, the location is *Documents and Settings* > *All Users* > ***Shared Documents*** > *SQL Anywhere 10* > *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 10\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.

In some cases, 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, except Windows CE, SQL Anywhere programs, including the database server and administration utilities, 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 directory (the directory where the program executable file is located).
2. Related directories. Directories with the following paths relative to the program executable directory:
    - Parent of the executable directory.
    - A child of the parent directory named *scripts*.
3. The current working directory. When a program is started, it has a current working directory (the directory from which it is started). This directory is searched for required files.
4. The Location registry entry. When installing onto Windows, SQL Anywhere adds a Location registry entry. The indicated directory is searched, followed by:
    - A child named *scripts*

- ♦ A child with the operating system name (*win32*, *x64*, and so on)

5. 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.
6. The CLASSPATH directories. For Java files, directories listed in the CLASSPATH environment variable are searched to locate files.
7. The PATH directories. Directories in the system path and the user's path are searched to locate files.

Extensive file searching on Windows CE

On Windows CE, SQL Anywhere programs, including the database server and administration utilities, perform a more extensive search for required files, such as DLLs. In these cases, SQL Anywhere programs look for files in the following order:

1. The executable directory (the directory where the program executable file is located).
2. \ (the root directory).
3. The Location registry entry (the directory specified by the registry key *HKEY_LOCAL_MACHINE \SOFTWARE\Sybase\SQL Anywhere\10.0\Location*).
4. The *\Windows* directory.

Extensive file searching on Unix

On Unix, SQL Anywhere programs, including the database server and administration utilities, 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. Paths relative to server current working directory (*./*):
   a. *./res*
   b. *./scripts*
   c. *./tix*
   d. *./bin*
   e. *./lib*
   f. *./java*
   g. *./shared*
   h. *./../shared*

   > **Note**
   > All of these subdirectories are searched in all steps, except step 6.

2. The executable path.

3. The SYBASE environment variable.
4. The PATH environment variable.
5. The LIBPATH environment variable:
   - LD_LIBRARY_PATH on Linux and Solaris
   - SHLIB_PATH on HP-UX
   - LIBPATH on AIX
   - DYLD_LIBRARY_PATH on Mac OS X
6. One of the following:
   a. *install-dir/scripts* (for files of type *.sql*)

   b. *install-dir/java* (for files of type *.zip*)

   c. *install-dir/charsets/unicode* (for files of type *.uct*)
7. The SQL Anywhere installation directory (*install-dir*), where *install-dir* is a single directory specified by the SQLANY*x* environment variable if it has been defined.

Extensive file searching on NetWare

On NetWare, SQL Anywhere programs, including the database server and administration utilities, perform a more extensive search for required files. In these cases, SQL Anywhere programs look for files in the following order:

1. The executable directory (the directory where the program executable file is located).
2. The current working directory.
3. The sibling directory of the executable directory named *scripts* (for *.sql* files).
4. The sibling directory of the executable directory named *java* (for *.zip* or *.jar* files).
5. The parent directory of the executable directory.
6. The *SYS:SYSTEM* directory.
7. The *SYS:SYSTEM\scripts* directory (for *.sql* files).
8. The *SYS:SYSTEM\java* directory (for *.zip* or *.jar* files).

Registry and INI files

On Windows operating systems (except Windows CE), SQL Anywhere uses several registry settings. On Unix and NetWare, these settings are held in initialization files instead.

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.

☞ For more information, see "File Hiding utility (dbfhide)" on page 608.

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.

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 no matter which user is logged on; 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 IIS to work with such a web server.

In general, the use of local machine settings is recommended.

Registry structure

On Windows (except Windows CE), 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
         10.0
      Sybase Central
         5.0.0
```

Modifying your registry is dangerous
Modify your registry at your own risk. It is recommended that you back up your system before modifying the registry.

Registry settings on installation

On Windows, the installation program makes the following registry settings in the *HKEY_LOCAL_MACHINE\Software\Sybase* registry:

- **SQL Anywhere\10.0\Location** This entry holds the installation directory location for the SQL Anywhere software. For example:

  ```
  Location "c:\Program Files\SQL Anywhere 10"
  ```

- **SQL Anywhere\10.0\Shared Location** This entry holds the installation directory location for shared software like Sybase Central and the Sun Java Runtime Environment (JRE). For example:

  ```
  Location "c:\Program Files\SQL Anywhere 10"
  ```

  This entry is used by Interactive SQL.

- **SQL Anywhere\10.0\Folder** This entry holds the name of the Start menu folder and is used by the SQL Anywhere installer. For example:

  ```
  Folder "SQL Anywhere 10"
  ```

- **SQL Anywhere\10.0\Online Resources** This entry holds the location for the Online Resources documentation. For example:

  ```
  Online Resources "c:\Program Files\SQL Anywhere 10\support\ianywhere.html"
  ```

- **SQL Anywhere\10.0\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. For a list of language codes, see “Understanding the locale language” on page 314.

- **SQL Anywhere\10.0\SNMPDLL\IniFile** This entry holds the location and name of the SNMP Extension Agent initialization file. For example:

  ```
  IniFile "c:\Program Files\SQL Anywhere 10\win32\sasnmp.ini"
  ```

- **SQL Anywhere\10.0\SNMPDLL\Pathname** This entry holds the location and name of the SNMP Extension Agent DLL. For example:

  ```
  Pathname "c:\Program Files\SQL Anywhere 10\win32\dbsnmp10.dll"
  ```

- **Sybase Central\5.0.0\Location** This entry holds the installation directory location for Sybase Central. For example:

  ```
  Location "c:\Program Files\SQL Anywhere 10\Sybase Central 5.0.0"
  ```

  This entry is used by Sybase Central.

- **Sybase Central\5.0.0\Shared Location** This entry holds the installation directory location for shared software like Sybase Central and the Sun Java Runtime Environment (JRE). For example:

  ```
  Location "c:\Program Files\SQL Anywhere 10\Sybase Central 5.0.0"
  ```

  This entry is used by Sybase Central.

- **Sybase Central\5.0.0\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. For a list of language codes, see "Understanding the locale language" on page 314.

Registry settings on Windows CE

You can specify which directory you want to use as the server's temporary directory on Windows CE by setting the following value in the registry:

HKEY_CURRENT_USER\Software\Sybase\SQL Anywhere\10.0\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 server will:

- use the *\Temp* directory if it exists.
- attempt 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.

CHAPTER 10

International Languages and Character Sets

Contents

About this chapter

This chapter describes how to configure your SQL Anywhere installation to handle international language issues.

Introduction to international languages and character sets

This section provides an introduction to the issues you may face when working in an environment that uses more than one character set, or when using languages other than English.

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 is localized to, and can be purchased in, one of five languages:

- English
- French
- German
- Japanese
- Simplified Chinese

Language choice is determined at installation. The software and online documentation is installed and configured for that language.

On Windows, the Start menu items allow the software to be reconfigured between the installed language and English. The Language Selection utility program (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 305, and "The Language Selection utility" on page 628.

The following table shows the availability of each language by operating system platform.

Platform	English	German	French	Japanese	Simplified Chinese
Windows	Yes	Yes	Yes	Yes	Yes
Windows CE	Yes	Yes	Yes	Yes	Yes
Linux	Yes	Yes		Yes	Yes
Unix	Yes				
Mac OS X	Yes				
NetWare	Yes				

Full software and documentation localization

SQL Anywhere for Windows is available in five languages, suitable for development, deployment and administration. The languages are:

- English
- French
- German
- Japanese
- Simplified Chinese

For these languages, all SQL Anywhere components are localized, including:

- Packaging
- Installer
- Documentation, including HTML Help and PDFs
- Software
  - Start menu items and program folders
  - 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

The following components are not localized and are only available in English:

- DataWindow .NET
- InfoMaker
- 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

SQL Anywhere international features

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 307.

  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 317.

  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*].

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 [database]" on page 456.

- **Character set conversion** SQL Anywhere converts data between the character set encoding on your server and client systems, thus maintaining the integrity of your data, even in mixed character set environments. See "Understanding character set conversion" on page 322.
- **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 [compatibility]" on page 406
  - "time_format option [compatibility]" on page 462
  - "timestamp_format option [compatibility]" on page 463
  - "CONVERT function [Data type conversion]" [*SQL Anywhere Server - SQL Reference*]

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 CE)

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).

ICU is needed when...	Notes
The client and database character sets are different, and when either of them is multibyte (including UTF-8). This includes Unicode ODBC, OLE DB, ADO.NET, and iAnywhere 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 bullet item above.	Character set conversion in the case of the third bullet item above 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 CE)

The following table specifies when and why ICU is needed for Windows CE:

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, ICU is recommended to improve character set conversion if you are using NCHAR, or if the CHAR character set is multibyte.

Note

ICU is not used for character set conversion on Windows CE. Because of this, the database character set must be the same as the operating system character set on the device, or it must be UTF-8.

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, provided that 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 CE)

For Unicode client applications, you will likely 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.

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 317
How are characters represented in software, and in SQL Anywhere in particular?	"Understanding character sets in software" on page 310
What collations does SQL Anywhere provide?	"Choosing collations" on page 320
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?	"Understanding character set conversion" on page 322
What character sets can I use for connection strings?	"Connection strings and character sets" on page 322
How do I change the collation sequence of an existing database?	"Changing a database from one collation to another" on page 326

Understanding character sets in software

This section provides general information about software issues related to international languages and character sets.

Overview of character sets, encodings, and collations

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.

For the purposes of 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.

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 of 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 of 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, as well as 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 on the multibyte character sets and collations, see "SQL Anywhere Collation Algorithm (SACA)" on page 317.

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 the console or command prompt window) in Windows XP/200x 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 328.

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.

☞ For more information on the CHAR and NCHAR data types, see "Character data types" [*SQL Anywhere Server - SQL Reference*].

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 332.

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, as well as 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:

- **Language** The language is a two-character string using the ISO-639 standard values: DE for German, FR for French, and so on. 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 314.

- **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 316.

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 323.

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 284.

2. In the case of Windows, if the SALANG environment variable doesn't exist, check the SQL Anywhere language registry entry, as described in "Registry settings on installation" on page 300.

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 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
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

Language	ISO_639 language code	Language label	Alternative label
Turkish	TR	turkish	N/A
Ukrainian	UK	ukrainian	N/A

Understanding the locale character set

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 282.

   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 connection parameter [CS]" on page 210.

3. 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 323.

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 as to which collations to use under certain circumstances.

☞ For more information about how to create a database with a specific collation, see “Creating a database with a named collation” on page 325 and “The Initialization utility” on page 614.

SQL Anywhere Collation Algorithm (SACA)

The SQL Anywhere Collation Algorithm provides reasonable comparison, ordering, and case conversion of single-byte and multibyte character sets. Each character or group of characters in the character set is mapped to one of up to 256 distinct values. All characters mapped to the same value are equal and sort together.

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 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.

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 the tailoring the UCA. ICU tailoring syntax is not supported in SQL Anywhere.

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, the Unicode Collation Algorithm 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
nöel	noel	none, diaresis, none, none	lower, lower, lower, lower
Noel	noel	none, none, none, none	upper, lower, lower, lower
Nöel	noel	none, diaresis, none, 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 nöel Nöel noel	♦ Accents ignored ♦ Case ignored ♦ All values considered equal ♦ Random order within set of four
Y	N	Noel noel nöel Nöel	♦ No-accents before accents, so o before ö ♦ Case ignored, uppercase and lowercase N are random within each set of two
N	Y	Noel Nöel nöel noel	♦ Uppercase before lowercase ♦ Accents ignored, o and ö are in random order within each set of two
Y	Y	Noel noel Nöel nöel	♦ No-accents before accents ♦ Uppercase before lowercase

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 or they can use the Unicode Collation Algorithm. 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 or can use the UTF8BIN collation, which uses the SQL Anywhere Collation Algorithm.

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.

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.

Choosing collations

When you create a database, SQL Anywhere can choose a default collation based on operating system language and character set settings. In most cases, 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. In some cases, SQL Anywhere supports 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.

☞ For more information about sorting of data and international features, see “SQL Anywhere international features” on page 306.

How SQL Anywhere chooses the 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 284.

- The character set comes from the SACHARSET environment variable (if it exists) or the operating system. See "SACHARSET environment variable" on page 282.

Considerations when choosing a collation

When choosing the collation for your database, consider the following:

- 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.
- There is a performance cost, as well as 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. In the case of Windows operating systems on the client computer, choose the ANSI character set.

Understanding 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.

This section describes how SQL Anywhere performs character set conversion. This information is provided for advanced users, such as those who may be deploying applications or databases in a multi-character-set environment.

SQL Anywhere implements character set conversion using the International Components for Unicode (ICU) open source library, developed and maintained by IBM.

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 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.
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.

   For maximum compatibility among different computers, you should use alpha-numeric characters (ASCII characters 32 to 126) 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.

International language and character set tasks

This section groups together the tasks associated with international language and character set issues.

Determining the default collation

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. Connect to the sample database.
2. Enter the following query:

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

   The default collation is returned.

   ☞ For more information about this collation, see "Choosing collations" on page 320.

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 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' );`	Language used by server console.
`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 323.

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 list of valid languages. See "Language label values" on page 315

   ☞ For more information on how to set environment variables on different operating systems, see "SQL Anywhere Environment Variables" on page 277.

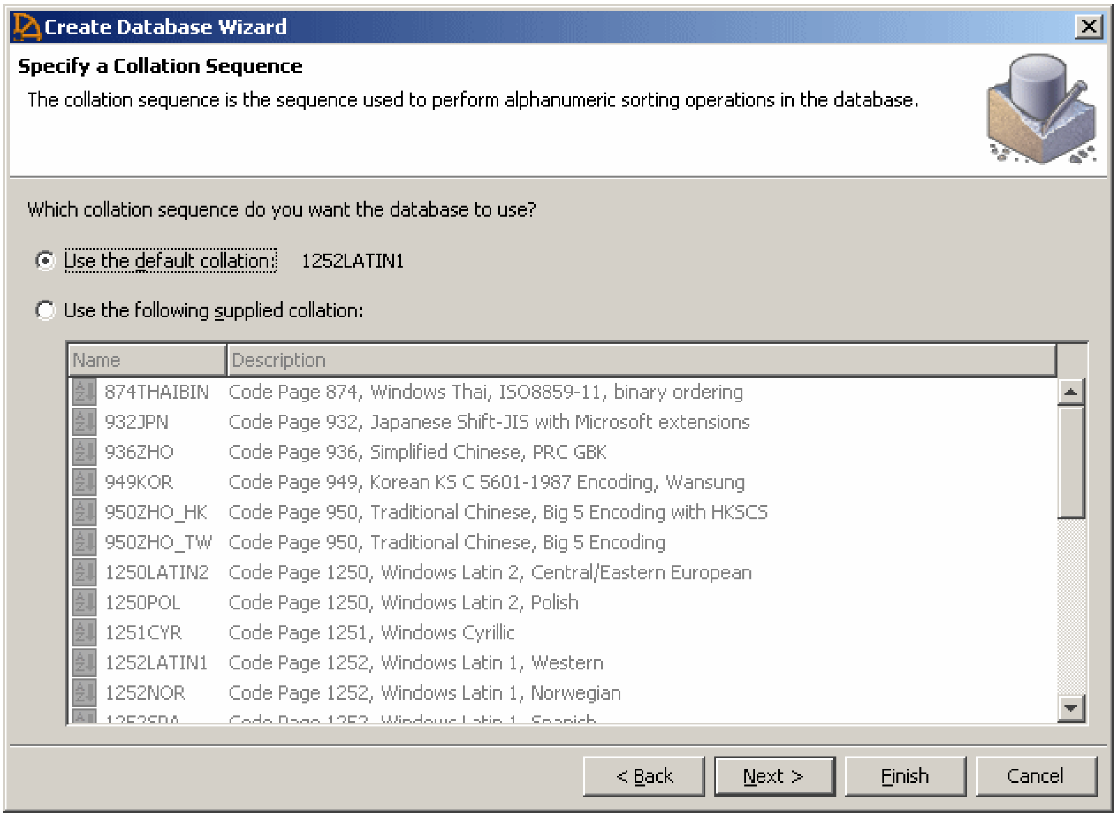
Creating a database with a named collation

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 320.

♦ To specify a database collation when creating a database (Sybase Central)

- You can use the Create Database wizard in Sybase Central to create a database. The wizard has a page where you choose a collation from a list.



Create Database Wizard

Specify a Collation Sequence

The collation sequence is the sequence used to perform alphanumeric sorting operations in the database.

Which collation sequence do you want the database to use?

● Use the default collation: 1252LATIN1

○ Use the following supplied collation:

Name	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
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, Windows 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

♦ To specify a database collation when creating a database (command prompt)

1. List the recommended collation sequences by typing the following at a command prompt:

```
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
```

♦ 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'
  ```

Changing 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 DATABASE_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.

2. Determine the character set for the data in the existing database.

   This should be the same as the database character set, but 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 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 server's character set conversion will automatically convert the data 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 TABLE statement” [*SQL Anywhere Server - SQL Reference*]
- “INPUT statement [Interactive SQL]” [*SQL Anywhere Server - SQL Reference*]
- “Creating a database with a named collation” on page 325
- “The Unload utility” on page 689
- “Rebuilding databases” [*SQL Anywhere Server - SQL Usage*]
- “CharSet connection parameter [CS]” on page 210

Character set and collation reference information

The following sections provide information about character sets and collations for SQL Anywhere.

Supported character sets

SQL Anywhere supports a growing list of hundreds of character sets and labels. The fastest way to determine if a certain character set or label is supported is to test it using the CSCONVERT function. The CSCONVERT function can convert a string from the database character set into the specified character set or label. If the character set or label is not supported, the database server returns an error. If the character set or label is supported, the database server returns a result set.

For example, to determine if the greek8 character set is supported, execute the following command in Interactive SQL:

```
SELECT CSCONVERT ( 'teststring', 'greek8' );
```

The database server returns a result value; this indicates that greek8 is supported.

☞ For more information on how to use the CSCONVERT function, see "CSCONVERT function [String]" [*SQL Anywhere Server - SQL Reference*].

Supported and alternate collations

Following is the list of supported CHAR collations in SQL Anywhere. This list can also be obtained by executing the following command at a command prompt:

```
dbinit -l
```

Collation label	Description
874THAIBIN	Code Page 874, Windows Thai, ISO88959-11, 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

Collation label	Description
1252LATIN1	Code Page 1252, Windows Latin 1, Western
1252NOR	Code Page 1252, Windows Latin 1, Norwegian
1252SPA	Code Page 1252, Windows Latin 1, Spanish
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, execute the following command at a command prompt:

```
dbinit -l+
```

Recommended character sets and collations

While SQL Anywhere recognizes the names of hundreds of character sets, code pages, encodings and collations, this section provides listings of those that are recommended for use with Windows and Unix platforms, depending on the language in use.

Note
For languages not found in the tables below, the UTF-8 encoding should be used with collations UCA or UTF8BIN.

Windows platforms

Language	Code page	Collation	Alternate collation
Arabic	cp1256	1256ARA	
Central and Eastern European	cp1250	1250LATIN2	
Danish	cp1252	1252LATIN1	
Dutch	cp1252	1252LATIN1	
English	cp1252	1252LATIN1	
Finnish	cp1252	1252SWEFIN	
French	cp1252	1252LATIN1	
German	cp1252	1252LATIN1	
Greek	cp1253	1253ELL	
Hebrew	cp1255	1255HEB	
Italian	cp1252	1252LATIN1	
Japanese	cp932	932JPN	
Korean	cp949	949KOR	
Lithuanian	cp1257	1257LIT	
Norwegian	cp1252	1252NOR	
Polish	cp1250	1250POL	
Portuguese	cp1252	1252LATIN1	
Russian	cp1251	1251CYR	
Simplified Chinese	cp936	936ZHO	

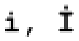
Language	Code page	Collation	Alternate collation
Spanish	cp1252	1252SPA	
Swedish	cp1252	1252SWEFIN	
Thai	cp874	874THAIBIN	
Traditional Chinese - Hong Kong	cp950	950ZHO_HK	
Traditional Chinese - Taiwan	cp950	950ZHO_TW	
Turkish	cp1254	1254TRK	1254TRKALT
Ukrainian	cp1251	1251CYR	
Western European	cp1252	1252LATIN1	

Unix platforms

Language	Character set	Collation	Alternate collation
Danish	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Dutch	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
English	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Finnish	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
French	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
German	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Italian	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Japanese	EUC-JP	EUC_JAPAN	
Korean	EUC-KR	EUC_KOREA	
Norwegian	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Portuguese	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Simplified Chinese	GB2312	EUC_CHINA	
Spanish	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Swedish	ISO-8859-15	ISO9LATIN1	ISO1LATIN1
Traditional Chinese	Big5	EUC_TAIWAN	
Western European	ISO-8859-15	ISO9LATIN1	ISO1LATIN1

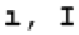
Turkish character sets and collations

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 lower-case equivalent of `I` is:

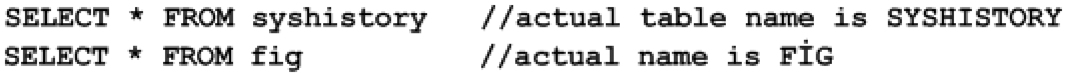
```
ı
```

However, the ANSI standard says that it is:

```
i
```

For this reason, correct case-insensitive matching is dependent on whether or not 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.

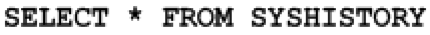
For example, consider the following statements, executed against a database using the 1254TRK collation:



```
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, and 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.

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 lower-case 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. While 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, in some cases it may be more expedient to make a decision 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
```

This is not correct for a Turkish user, but may be acceptable in some cases.

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.

CHAPTER 11

Managing User IDs and Permissions

Contents

About this chapter

Each user of a database must have a name they enter when connecting to the database, called a user ID. This chapter describes how to manage user IDs.

Database permissions overview

Proper management of user IDs and permissions lets users of a database perform their jobs effectively, while maintaining the security and privacy of information within the database.

Database permissions are assigned to user IDs. Throughout this chapter, the term **user** is used as a synonym for user ID. You use SQL statements for assigning user IDs to new users of a database, granting and revoking permissions for database users, and finding out the current permissions of users.

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.

DBA authority overview

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 614.

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.

Users with DBA authority

A user with DBA authority becomes the **database administrator**. In this chapter, 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 chapter 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.

Note
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.

RESOURCE authority overview

RESOURCE authority is the permission to create database objects, such as tables, views, stored procedures, and triggers. RESOURCE authority may be granted only by the DBA.

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.

BACKUP authority overview

BACKUP authority is the permission to back up databases and transaction logs with archive or image backups using the BACKUP statement or dbbackup utility. BACKUP authority can be granted only by the DBA. See “BACKUP statement” [*SQL Anywhere Server - SQL Reference*] and “The Backup utility” on page 590.

VALIDATE authority overview

VALIDATE authority is the permission to perform database, table, index, and checksum validation using the VALIDATE statement or dbvalid utility. VALIDATE authority can be granted only by the DBA. See “VALIDATE statement” [*SQL Anywhere Server - SQL Reference*] and “The Validation utility” on page 704.

Ownership permissions overview

The creator of a database object becomes the owner of that object. Ownership of a database object carries with it permissions to perform actions on that object. These permissions are not assigned to users in the same way that other permissions in this chapter are assigned.

Owners

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.

The DBA has permission to modify any component within the database, and so could delete a table created by another user. The DBA has all the permissions regarding database objects that the owners of each object have. As well, the DBA 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. A use for this ability is discussed in "Groups without passwords" on page 356. Despite this possibility, this chapter refers interchangeably to the owner and creator of database objects as the same user.

Table and views permissions overview

There are several distinct permissions you may grant to user IDs concerning tables and views:

Permission	Description
ALTER	Permission to alter the structure of a table or create a trigger on a table.
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.

Group permissions overview

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.

You can assign permissions to a group in exactly the same way as to an individual user. You can then assign membership in appropriate groups to each new user of the database, and they gain a set of permissions by virtue of their group membership.

Example

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.

Each user ID can be a member of multiple groups, and they inherit all permissions from each of the groups.

Managing individual user IDs and permissions

This section describes how to create new users and grant permissions 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.

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 GRANT CONNECT 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 it to other groups

  For more information, see "Granting group membership to existing users or groups" on page 353.

- set its permissions on tables, views, and procedures

  For more information, see "Managing individual user IDs and permissions" on page 340.

- set it as the publisher or as a remote user of the database

  For more information, see "Managing SQL Remote permissions" [*SQL Remote*].

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.

The DBA can set the permissions granted automatically to new users by assigning permissions to the special PUBLIC user group, as discussed in "Special groups" on page 357.

Restrictions on user IDs and passwords

When creating user IDs and passwords, the following restrictions apply. They cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons

♦ To create a new user (Sybase Central)

1. Open the Users & Groups folder.
2. From the File menu, choose New ► User.

   The Create User wizard appears.
3. Follow the instructions in the wizard.

♦ To create a new user (SQL)

1. Connect to a database as a user with DBA authority.
2. Execute a GRANT CONNECT TO statement.

Example

Add a new user to the database with the user ID of M_Haneef and a password of Welcome.

```
GRANT CONNECT TO M_Haneef
IDENTIFIED BY Welcome;
```

See also

♦ "GRANT statement" [*SQL Anywhere Server - SQL Reference*]

Setting a password

A user must have a password to be able to connect to the database. Passwords cannot be more than 255 bytes in length, and cannot contain semicolons, leading spaces, or trailing spaces. Passwords are case sensitive.

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. 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.

Changing a password

Using the GRANT statement, you can change your password or that of another user if you have DBA authority. For example, the following command changes the password for user ID M_Haneef to new-password:

```
GRANT CONNECT TO M_Haneef
IDENTIFIED BY new-password;
```

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. The following command changes the password for the DBA user to new_password:

```
GRANT CONNECT TO DBA
IDENTIFIED BY new_password;
```

Setting user and group options

In Sybase Central, configurable options for users and groups are located in the User Options and Group Options dialogs (the same dialog 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. Connect to a database and open the Users & Groups folder.
2. Select the desired user or group and then choose File ► Options.
3. Edit the desired values.
4. Click Set Permanent Now.

♦ To set the options for a user or group (SQL)

- Specify the desired properties within a SET OPTION statement.

See also

♦ “Setting properties for database objects” [*SQL Anywhere Server - SQL Usage*]

♦ “Database Options” on page 371

Granting DBA and RESOURCE authority

You can grant DBA and RESOURCE authority in the same manner.

♦ To grant RESOURCE authority to a user ID

1. Connect to the database as the DBA.
2. Execute the SQL statement:

   ```
   GRANT RESOURCE TO user-id;
   ```

For DBA authority, the appropriate SQL statement is:

```
GRANT DBA TO user-id;
```

Notes

- Only the DBA can grant DBA or RESOURCE authority to database users.
- DBA authority is very powerful: anyone with this authority has the ability to perform any action on the database, as well as access to all the information in the database. It is wise to grant DBA authority to only a few people.
- Consider giving users who need DBA authority two user IDs, one with DBA authority and one without, so that they connect as DBA only when necessary.
- RESOURCE authority allows the user to create new database objects, such as tables, views, indexes, procedures, or triggers.

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. Connect to the database.
2. Open the Tables folder for that database.
3. Select a table and then choose File ► Properties.
4. On the Permissions tab of the Table property sheet, configure the permissions for the table:

    - Click Grant to select users or groups to which to grant permissions.
    - Click the fields beside the user or group to set specific permissions. Permissions are indicated by a check mark, and grant options are indicated by a check mark with two '+' signs.

- Select a user and click Change beside References, Select, or Update to set that type of permission on individual columns.
- Select a user or group in the list and click Revoke to revoke all permissions.

Tips
You can also assign permissions from the User or Group property sheet. To assign permissions to many users and groups at once, use the table's property sheet. To assign permissions to many tables at once, use the User property sheet.

♦ To grant permissions on tables or columns (SQL)

1. Connect to the database as the DBA or as the owner of the table.
2. Execute a GRANT statement to assign the permission.

☞ For more information, 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 the DBA, 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 the DBA, 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 343.

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 DBA.

♦ 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 DBA.

  ♦ a non-DBA, but also the owner of all the base table(s) referred to by the view.

  ♦ a non-DBA, and not the owner of some or all of 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.

♦ To grant permissions on views (Sybase Central)

1. Connect to the database.

2. Open the Views folder for that database.

3. Select a view and then choose File ► Properties.

4. On the Permissions tab of the View property sheet, configure the permissions for the view:

   ♦ Click Grant to select users or groups to which to grant full permissions.

   ♦ Click in the fields beside the user or group to set specific permissions. Permissions are indicated by a check mark, and grant options are indicated by a check mark with two '+' signs.

   ♦ Select a user or group in the list and click Revoke to revoke all permissions.

Tip
You can also assign permissions from the User or Group property sheet. To assign permissions to many users and groups at once, use the View property sheet. To assign permissions to many views at once, use the User or Group property sheet.

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 in “Granting permissions on tables” on page 343 with the WITH GRANT OPTION. This option gives the right to pass on the permission to other users. In the context of groups, you can read about this feature in section “Permissions of groups” on page 355.

In Sybase Central, you can specify a grant option by displaying the property sheet of a user, group, or table, clicking the Permissions tab, and double-clicking in 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 the DBA, or as the owner of sample_table

2. Execute the SQL statement:

```
GRANT DELETE ON sample_table
TO M_Haneef
WITH GRANT OPTION;
```

☞ For more information, see “GRANT statement” [*SQL Anywhere Server - SQL Reference*]

Granting permissions on procedures

The DBA 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, discussed in “Granting permissions on tables” on page 343. 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. Connect to the database.

2. Open the Procedures & Functions folder for that database.

3. Select a procedure and then choose File ▶ Properties.

4. On the Permissions tab of the Procedure property sheet, configure the permissions for the procedure:

   - Click Grant to select users or groups to which to grant all permissions on the procedure.
   - Click beside users in the Execute column to toggle between granting or not granting permission.
   - Select a user or group in the list and click Revoke to revoke all permissions.

> **Tip**
> You can also assign permissions from the User or Group property sheet. To assign permissions to many users and groups at once, use the Procedure property sheet. To assign permissions to many procedures at once, use the User or Group property sheet.

♦ To grant permissions on procedures (SQL)

1. Connect to the database as the DBA 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 the DBA 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 will execute 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 or not 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*]

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.

☞ For more information on trigger permissions, 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.

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 grant remote permissions to users (Sybase Central)

1. Connect to a database.
2. Open the Users & Groups folder.
3. Select the desired user and then choose File ► Change to Remote User.

   The Change User to Remote User dialog appears.
4. In the resulting dialog, enter the desired values.

Once you have granted remote permissions to a user, you can subscribe it to publications:

♦ To revoke remote permissions from remote users

1. Open either the Users & Groups folder or the SQL Remote Users folder.
2. Select the desired remote user and then choose File ► Revoke Remote.

☞ For more information, see “SQL Remote Concepts” [*SQL Remote*].

Revoking user permissions

Any 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.

The REVOKE statement is the exact opposite of the GRANT statement. To disallow M_Haneef from executing my_procedure, the command is:

```
REVOKE EXECUTE
ON my_procedure
FROM M_Haneef;
```

The DBA or the owner of the procedure must issue this command.

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.

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.

Permission to delete rows from sample_table may be revoked by issuing the following command:

```
REVOKE DELETE
ON sample_table
FROM M_Haneef;
```

See also

- ♦ “Granting permissions on tables” on page 343
- ♦ “Granting permissions on views” on page 345
- ♦ “Granting permissions on procedures” on page 346

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.

Only the DBA can delete a user.

♦ To delete a user from the database (Sybase Central)

1. Open the Users & Groups folder.

2. Select the desired user and then choose File ► Delete.

♦ To delete a user from the database (SQL)

- Execute a REVOKE CONNECT FROM statement.

Example

Remove the user M_Haneef from the database.

```
REVOKE CONNECT FROM M_Haneef;
```

See also

- “REVOKE statement” [*SQL Anywhere Server - SQL Reference*]
- “Revoking user permissions” on page 349
- “Deleting groups from the database” on page 357

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

- Select the database in the left pane, and click the Connected Users tab in the right pane.

  This tab displays all users currently connected to a given database (including yourself), regardless of the application that they used to connect (Sybase Central, Interactive SQL, a custom client application, and so on).

♦ To inspect the properties of a user's connection to a database

1. Select the database in the left pane, and click the Connected Users tab in the right pane.
2. Select the desired user and then choose File ► Properties.
3. Inspect the desired properties.

♦ To disconnect users from a database

1. Select the database in the left pane, and click the Connected Users tab in the right pane.
2. Select the desired user and then choose File ► Disconnect.

Managing groups

Once you understand how to manage permissions for individual users (as described in the previous section), working with groups is straightforward. A group is identified by a user ID, just as a single user is. However, a group user ID has the permission to have members.

Inheriting authorities

When you grant permissions to a group or revoke permissions from a group for tables, views, and procedures, all members of the group inherit those changes. The following authorities are not inherited; you must assign them individually to each of the individual user IDs requiring them:

- DBA
- BACKUP
- VALIDATE
- GROUP
- RESOURCE
- REMOTE
- REMOTE DBA
- CONSOLIDATE
- PUBLISH

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.

A group is simply a user ID with special permissions. You grant permissions to a group and revoke permissions from a group in exactly the same manner as any other user, using the commands described in "Managing individual user IDs and permissions" on page 340.

You can construct a hierarchy of groups, where each group inherits permissions from its parent group. That means that a group can also be a member of a group. As well, each user ID may belong to more than one group, so the user-to-group relationship is many-to-many.

The ability to create a group without a password enables you to prevent anybody from connecting to the database using the group user ID.

☞ For more information about this security feature, see "Groups without passwords" on page 356.

☞ For more information on altering database object properties, see "Setting properties for database objects" [*SQL Anywhere Server - SQL Usage*].

☞ For more information about granting remote permissions for groups, see "Granting and revoking remote permissions" on page 348.

Creating groups

You can create a new group in both Sybase Central and Interactive SQL. You need DBA authority to create a new group.

♦ To create a new group (Sybase Central)

1. Click the Users & Groups folder.
2. From the File menu, choose New ▶ Group.

   The Create Group wizard appears.
3. Follow the instructions in the wizard.

♦ To create a new group (SQL)

- Execute a GRANT GROUP TO statement. If the user ID you specify in this statement has not already been created, the statement fails.

Example

Create the user ID personnel.

```
GRANT CONNECT
TO 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*]
- ♦ “Creating new users” on page 340

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.

Only the DBA can grant membership in a group.

♦ To add a user or group to another group (Sybase Central)

1. Open the Users & Groups folder.

2. Select the user/group that you want to add to a group, and from the File menu choose New ▶ Memberships.

   The New Memberships dialog appears.

3. Select the desired group and click OK.

4. The group membership appears on the Memberships tab in the right pane for the selected user/group.

♦ To add a user or group to another group (SQL)

- Execute a GRANT MEMBERSHIP IN GROUP statement, specifying the desired 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

- ♦ "GRANT statement" [*SQL Anywhere Server - SQL Reference*]
- ♦ "Creating new users" on page 340

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 the DBA 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. Open the Users & Groups folder.

2. Select the desired user/group and click the Memberships tab in the right pane.

3. Select the desired group, and then choose File ▶ Remove Memberships.

   The user or group is then removed from this desired group.

Tip
You can perform this action by selecting the group, clicking the Members tab in the right pane, and then selecting the user/group you want to remove and choosing File ► Remove Members from the popup menu.

♦ To remove a user or group from another group (SQL)

- Execute a REVOKE MEMBERSHIP IN GROUP statement, specifying the desired group and the users involved.

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 340
- ♦ "Deleting users from the database" on page 349
- ♦ "Deleting groups from the database" on page 357

Permissions of groups

You can grant permissions to groups in exactly the same way as to any other user ID. Permissions on tables, views, and procedures are inherited by members of the group, including other groups and their members. Some complexities to group permissions exist that database administrators need to keep in mind.

Notes

Members of a group do not inherit the DBA, RESOURCE, and GROUP permissions. Even if the personnel user ID has RESOURCE authority, the members of personnel do not have RESOURCE authority.

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, as well as 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 the DBA 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.

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.

☞ For an example, see the section "Database object names and prefixes" on page 359.

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 GRANT CONNECT statement without a password. The following statement creates a user ID personnel:

```
GRANT CONNECT
TO 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 the DBA can perform this assignment of ownership.

Special groups

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.

The SYS group

The SYS group owns the system tables and views for the database, which contain the full description of database structure, 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 the chapters "Tables" [*SQL Anywhere Server - SQL Reference*], and "System views" [*SQL Anywhere Server - SQL Reference*].

The 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 the DBA. You can also REVOKE membership in PUBLIC for users if you want.

The 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.

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 the DBA can delete a group.

♦ To delete a group from the database (Sybase Central)

1. Open the Users & Groups folder.
2. Select the desired group and then choose File ▶ Delete.

♦ To delete a group from the database (SQL)

1. Connect to a database.
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” on page 349
- “Deleting users from the database” on page 349

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 guide, 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 sufficient 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.

```
GRANT CONNECT TO 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.

```
GRANT CONNECT TO Sally IDENTIFIED BY xxxxx;
GRANT CONNECT TO Joe IDENTIFIED BY xxxxx;
GRANT CONNECT TO 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.) Thus, 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.
- 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.

☞ For more information about how to create views, see "Working with views" [*SQL Anywhere Server - SQL Usage*].

☞ For more information about view permissions, see "Granting permissions on views" on page 345.

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 in terms of rows or in terms of 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

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 the DBA, enter the following:

```
CONNECT DBA
IDENTIFIED by sql;

GRANT CONNECT
TO SalesManager
IDENTIFIED BY sales;
```

2. Define a view which 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.

☞ For more information about allowing data modification on views, see “Using views” [*SQL Anywhere Server - SQL Usage*].

Using procedures for tailored security

While views restrict access on the basis of data, procedures restrict the actions a user may take. As described in “Granting permissions on procedures” on page 346, a user may have EXECUTE permission on a procedure without having any permissions on the table or tables on which the procedure acts.

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 the manner in which data in the database can be modified.

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 of 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.

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 the DBA 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*]

☞ For more information about options, see “Database Options” on page 371.

☞ For information on the SET OPTION statement, see “SET OPTION statement” [*SQL Anywhere Server - SQL Reference*].

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.
- **max_statement_count** Limits the number of prepared statements for a connection.
- **background_priority** Limits the impact requests on the current connection have on the performance of other connections.

Database option settings are not inherited through the group structure.

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.

The DBA has 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 the DBA has access to the SYS.SYSUSERPERM view, which contains all information about the permissions of users of the database, as well as 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 of 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*].
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*].

CHAPTER 12

Database Options

Contents

About this chapter

This chapter describes the database, replication, compatibility, and Interactive SQL options that can be set to customize and modify database behavior.

Introduction to 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/2003 generates errors.
- **Error handling** You can control what happens when errors such as dividing by zero, or overflow errors, occur.
- **Concurrency and transactions** You can control the degree of concurrency, and details of COMMIT behavior.

Setting 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*]

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 login_mode = Integrated;
```

The following statement applies a change to the PUBLIC user ID, a user group to which all users belong.

```
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.

☞ For more information, see “SET OPTION statement” [*SQL Anywhere Server - SQL Reference*].

Note
In databases that use a Turkish collation or are case sensitive, option names referenced in queries should be written using the case specified in "Alphabetical list of options" on page 389.
In either of these situations, 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';
```

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 permissions. 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.

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.

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 XP/ 200x 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.

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

- Current option settings for your connection are 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;
  ```

  If you want to filter the result or order by anything other than name, you could also use a SELECT statement. For example:

  ```
  SELECT *
  FROM sa_conn_properties()
  WHERE PropDescription LIKE '%cache%'
  ORDER BY PropNum;
  ```

  For more information, see “sa_conn_properties system procedure” [*SQL Anywhere Server - SQL Reference*].

- In Interactive SQL, the SET statement with no arguments lists the current setting of options.

  ```
  SET;
  ```

- In Sybase Central, select a database, and then choose File ▶ Options.

- Use the following query on the SYSOPTIONS system view:

  ```
  SELECT *
  FROM SYSOPTIONS;
  ```

  This displays all PUBLIC values, and those USER values that have been explicitly set.

Getting individual option values

You can obtain a single setting using the CONNECTION_PROPERTY system function. For example, the following statement reports the value of the ansi_integer_overflow option:

```
SELECT CONNECTION_PROPERTY ('ansi_integer_overflow');
```

☞ For more information, see “CONNECTION_PROPERTY function [System]” [*SQL Anywhere Server - SQL Reference*].

Initial option settings

Connections to SQL Anywhere can be made through the TDS protocol (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. As it is shipped, SQL Anywhere uses this method to set 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.

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.

☞ For more information, see “login_procedure option [database]” on page 424, “sp_login_environment system procedure” [*SQL Anywhere Server - SQL Reference*], and “sp_tsql_environment system procedure” [*SQL Anywhere Server - SQL Reference*].

Deleting option settings

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.

For example, the following statement resets the ansi_integer_overflow option to its default value:

```
SET OPTION ansi_integer_overflow =;
```

☞ For more information, 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 377

- “Compatibility options” on page 382
- “SQL Remote options” on page 387
- “Interactive SQL options” on page 559

Database options

This section lists all database options.

Option	Values	Default
"allow_snapshot_isolation option [database]" on page 389	On, Off	Off
"auditing option [database]" on page 395	On, Off	Off
"auditing_options option [database]" on page 396	Reserved	Reserved
"background_priority option [database]" on page 397	On, Off	Off
"blocking option [database]" on page 398	On, Off	On
"blocking_timeout option [database]" on page 398	Integer (in milliseconds)	0
"checkpoint_time option [database]" on page 399	Number of minutes	60
"cis_option option [database]" on page 400	0, 7	0
"cis_rowset_size option [database]" on page 400	Integer	50
"collect_statistics_on_dml_updates option [database]" on page 401	On, Off	On
"conn_auditing option [database]" on page 403	On, Off	On
"cooperative_commit_timeout option [database]" on page 404	Integer (in milliseconds)	250
"cooperative_commits option [database]" on page 405	On, Off	On
"debug_messages option [database]" on page 408	On, Off	Off
"dedicated_task option [database]" on page 409	On, Off	Off
"default_dbspace option [database]" on page 409	String	Empty string (use the system dbspace)
"default_timestamp_increment option [database] [MobiLink client]" on page 410	Integer (in microseconds from 1 to 60000000)	1
"delayed_commit_timeout option [database]" on page 410	Integer (in milliseconds)	500
"delayed_commits option [database]" on page 411	On, Off	Off

Option	Values	Default
“exclude_operators option [database]” on page 413	Reserved	Reserved
“extended_join_syntax option [database]” on page 413	On, Off	On
“first_day_of_week option [database]” on page 414	1, 2, 3, 4, 5, 6, 7	7 (Sunday is the first day of the week)
“for_xml_null_treatment option [database]” on page 416	Empty, Omit	Omit
“force_view_creation option [database]” on page 416	Reserved	Reserved
“global_database_id option [database]” on page 417	Integer	2147483647
“http_session_timeout option [database]” on page 417	Integer	30
“integrated_server_name option [database]” on page 418	String	NULL
“java_location option [database]” on page 420	String	Empty string
“java_main_userid option [database]” on page 420	String	Default DBA user
“java_vm_options option [database]” on page 421	String	Empty string
“log_deadlocks option [database]” on page 422	On, Off	Off
“login_mode option [database]” on page 422	Standard, Integrated, Kerberos, Mixed (deprecated)	Standard
“login_procedure option [database]” on page 424	String	sp_login_environment
“materialized_view_optimization option [database]” on page 426	Disabled, Fresh, Stale, *N* { Minute[s] | Hour[s] | Day[s] | Week[s] | Month [s] }	Stale
“max_cursor_count option [database]” on page 427	Integer	50
“max_plans_cached option [database]” on page 427	Integer	20

Option	Values	Default
“max_query_tasks option [database]” on page 428	Integer	0
“max_recursive_iterations option [database]” on page 429	Integer	100
“max_statement_count option [database]” on page 430	Integer >=0	50
“max_temp_space option [database]” on page 431	Integer [k|m|g|p]	0
“min_password_length option [database]” on page 432	Integer >=0	0 characters
“odbc_describe_binary_as_varbinary [database]” on page 434	On, Off	Off
“odbc_distinguish_char_and_varchar option [database]” on page 435	On, Off	Off
“oem_string option [database]” on page 435	String (up to 128 bytes)	Empty string
“on_charset_conversion_failure option [database]” on page 437	Ignore, Warning, Error	Ignore
“optimization_goal option [database]” on page 439	First-row or All-rows	All-rows
“optimization_level option [database]” on page 440	0–15	9
“optimization_workload option [database]” on page 441	Mixed, OLAP	Mixed
“pinned_cursor_percent_of_cache option [database]” on page 442	Integer, between 0–100	10
“post_login_procedure [database]” on page 443	String	Empty string
“precision option [database]” on page 444	Integer, between 1 and 127, inclusive	30
“prefetch option [database]” on page 445	Off, Conditional, Always	Conditional
“preserve_source_format option [database]” on page 446	On, Off	On
“prevent_article_pkey_update option [database] [MobiLink client]” on page 447	On, Off	On

Option	Values	Default
“read_past_deleted option [database]” on page 449	On, Off	On
“recovery_time option [database]” on page 449	Integer, in minutes	2
“remote_idle_timeout option [database]” on page 450	Integer, in seconds	15
“request_timeout option [database]” on page 452	Integer (0 through 86400, in seconds)	0
“return_date_time_as_string option [database]” on page 452	On, Off	Off
“rollback_on_deadlock [database]” on page 453	On, Off	On
“row_counts option [database]” on page 454	On, Off	Off
“scale option [database]” on page 455	Integer, between 0 and 127, inclusive, and less than the value specified for the precision database option	6
“secure_feature_key [database]” on page 455	String	NULL
“sort_collation option [database]” on page 456	Internal, collation_name, or collation_id	Internal
“subsume_row_locks option [database]” on page 459	On, Off	On
“suppress_tds_debugging option [database]” on page 459	On, Off	Off
“synchronize_mirror_on_commit option [database]” on page 460	On, Off	Off
“tds_empty_string_is_null option [database]” on page 460	On, Off	Off
“temp_space_limit_check option [database]” on page 461	On, Off	On

Option	Values	Default
"time_zone_adjustment option [database]" on page 462	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 server's time zone, depending on the client's connection type
"truncate_timestamp_values option [database] [MobiLink client]" on page 464	On, Off	Off
"truncate_with_auto_commit option [database]" on page 466	On, Off	On
"updatable_statement_isolation option [database]" on page 467	0, 1, 2, 3	0
"update_statistics option [database]" on page 468	On, Off	On
"user_estimates option [database]" on page 469	Enabled, Disabled, Override-Magic	Override-Magic
"uuid_has_hyphens option [database]" on page 469	On, Off	On
"verify_password_function option [database]" on page 470	String	Empty string
"wait_for_commit option [database]" on page 472	On, Off	Off
"webservice_namespace_host option [database]" on page 472	NULL, hostname-string	NULL

Compatibility options

The following options allow SQL Anywhere behavior to be made compatible with Adaptive Server Enterprise, or to support both old behavior and allow ISO SQL/2003 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.

☞ For a listing, 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 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.

☞ For information on how the settings are made, see “System procedures” [*SQL Anywhere Server - SQL Reference*].

Options for Open Client and JDBC connection compatibility with Adaptive Server Enterprise

Option	Setting
allow_nulls_by_default	Off
ansi_blanks	Off
ansinull	On
chained	Off
continue_after_raiserror	On
date_format	YYYY-MM-DD
date_order	MDY
escape_character	Off
float_as_double	On
isolation_level	1
on_tsql_error	Continue for jConnect connections
quoted_identifier	Off
time_format	HH:NN:SS.SSS

Option	Setting
timestamp_format	YYYY-MM-DD HH:NN:SS.SSS
tsql_hex_constant	On
tsql_outer_joins	Off
tsql_variables	On

Transact-SQL and SQL/2003 compatibility options

The following table lists the compatibility options, their allowed values, and their default settings.

Option	Values	Default
"allow_nulls_by_default option [compatibility]" on page 389	On, Off	On
"ansi_blanks option [compatibility]" on page 390	On, Off	Off
"ansi_close_cursors_on_rollback option [compatibility]" on page 391	On, Off	Off
"ansi_integer_overflow option [compatibility]" on page 391	On, Off	On
"ansi_permissions option [compatibility]" on page 392	On, Off	On
"ansi_substring option [compatibility]" on page 392	On, Off	On
"ansi_update_constraints option [compatibility]" on page 394	Off, Cursors, Strict	Cursors
"ansinull option [compatibility]" on page 394	On, Off	On
"automatic_timestamp option [compatibility]" on page 396	On, Off	Off
"chained option [compatibility]" on page 399	On, Off	On
"close_on_endtrans option [compatibility]" on page 401	On, Off	On
"continue_after_raiserror option [compatibility]" on page 403	On, Off	On
"conversion_error option [compatibility]" on page 404	On, Off	On

Option	Values	Default
“date_format option [compatibility]” on page 406	String	YYYY-MM-DD
“date_order option [compatibility]” on page 407	MDY, YMD, DMY	YMD
“divide_by_zero_error option [compatibility]” on page 412	On, Off	On
“escape_character option [compatibility]” on page 413	Reserved	Reserved
“fire_triggers option [compatibility]” on page 414	On, Off	On
“float_as_double option [compatibility]” on page 415	On, Off	Off
“isolation_level option [compatibility]” on page 419	0, 1, 2, 3	0
“nearest_century option [compatibility]” on page 433	Integer (between 0 and 100 inclusive)	50
“non_keywords option [compatibility]” on page 433	String (Comma-separated keywords list)	Empty string (No keywords turned off)
“on_tsql_error option [compatibility]” on page 438	Stop, Conditional, Continue	Conditional
“optimistic_wait_for_commit option [compatibility]” on page 438	On, Off	Off
“percent_as_comment option [compatibility]” on page 442	On, Off	On
“query_plan_on_open option [compatibility]” on page 447	On, Off	Off
“quoted_identifier option [compatibility]” on page 448	On, Off	On
“ri_trigger_time option [compatibility]” on page 453	Before, After	After
“sql_flagger_error_level option [compatibility]” on page 457	E, I, F, W	W
“sql_flagger_warning_level option [compatibility]” on page 457	E, I, F, W	W

Option	Values	Default
"string_rtruncation option [compatibility]" on page 458	On, Off	On
"time_format option [compatibility]" on page 462	String	HH:NN:SS.SSS
"timestamp_format option [compatibility]" on page 463	String	YYYY-MM-DD HH:NN:SS.SSS
"tsql_hex_constant option [compatibility]" on page 466	On, Off	On
"tsql_outer_joins option [compatibility]" on page 467	On, Off	Off
"tsql_variables option [compatibility]" on page 467	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 [database] [MobiLink client]” on page 410	Integer (in microseconds from 1 to 60000000)	1
“delete_old_logs option [MobiLink client] [SQL Remote] [Replication Agent]” on page 412	On, Off, Delay, *n* days	Off
“prevent_article_pkey_update option [database] [MobiLink client]” on page 447	On, Off	On
“truncate_timestamp_values option [database] [MobiLink client]” on page 464	On, Off	Off

SQL Remote options

The following options are included to provide control over SQL Remote replication behavior.

Option	Values	Default
"blob_threshold option [SQL Remote]" on page 397	Integer (in kilobytes)	256
"compression option [SQL Remote]" on page 402	Integer, from -1 to 9	6
"delete_old_logs option [MobiLink client] [SQL Remote] [Replication Agent]" on page 412	On, Off, Delay, *n* days	Off
"qualify_owners option [SQL Remote]" on page 447	On, Off	On
"quote_all_identifiers option [SQL Remote]" on page 448	On, Off	Off
"replication_error option [SQL Remote]" on page 451	Stored procedure name	(no procedure)
"replication_error_piece option [SQL Remote]" on page 451	Stored procedure name	(no procedure)
"subscribe_by_remote option [SQL Remote]" on page 458	On, Off	On
"verify_all_columns option [SQL Remote]" on page 470	On, Off	Off
"verify_threshold option [SQL Remote]" on page 472	Integer (in bytes)	1000

Replication Agent options

The following options are included to provide control over Replication Agent replication behavior.

Option	Values	Default
“delete_old_logs option [MobiLink client] [SQL Remote] [Replication Agent]” on page 412	On, Off, Delay, *n* days	Off
“replicate_all option [Replication Agent]” on page 450	On, Off	Off

Alphabetical list of options

This section lists options alphabetically.

allow_nulls_by_default option [compatibility]

Function

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 Open Client and jConnect connections

Description

The allow_nulls_by_default option is included for Transact-SQL compatibility. See “Setting options for Transact-SQL compatibility” [*SQL Anywhere Server - SQL Usage*].

allow_snapshot_isolation option [database]

Function

Controls whether snapshot isolation is enabled or disabled.

Allowed values

On, Off

Default

Off

Scope

Can be set for the PUBLIC group only. DBA authority is required to set this option.

Description

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 in the event that 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 [compatibility]" on page 419
- "updatable_statement_isolation option [database]" on page 467
- "Snapshot isolation" [*SQL Anywhere Server - SQL Usage*]
- "Isolation levels and consistency" [*SQL Anywhere Server - SQL Usage*]
- "Enabling snapshot isolation" [*SQL Anywhere Server - SQL Usage*]

Example

The following statement enables snapshot isolation for a database:

```
SET OPTION PUBLIC.allow_snapshot_isolation = 'On';
```

ansi_blanks option [compatibility]

Function

Controls behavior when character data is truncated at the client side.

Allowed values

On, Off

Default

Off

Description

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.

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.

ansi_close_cursors_on_rollback option [compatibility]

Function

Controls whether cursors that were opened WITH HOLD are closed when a ROLLBACK is performed.

Allowed values

On, Off

Default

Off

Description

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 [compatibility]” on page 401

ansi_integer_overflow option [compatibility]

Function

Controls what happens when an arithmetic expression causes an integer overflow error.

Allowed values

On, Off

Default

On

Description

The ISO SQL/2003 standard requires integer overflow should result in a `SQLSTATE = 22003 - overflow` error. SQL Anywhere behavior was previously different from this. The ansi_integer_overflow option can be set to Off to maintain compatibility with previous releases of the software.

ansi_permissions option [compatibility]

Function

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 is required to set this option.

Description

With ansi_permissions set to On, the SQL/2003 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.

ansi_substring option [compatibility]

Function

Controls the behavior of the SUBSTRING (SUBSTR) function when negative values are provided for the start or length parameters.

Allowed values

Off, On

Default

On

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Description

When the ansi_substring option is set to On, the behavior of the SUBSTRING function corresponds to ANSI/ ISO SQL/2003 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*]

Examples

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 [compatibility]

Function

Controls the range of updates that are permitted.

Allowed values

Off, Cursors, Strict

Default

Cursors

Description

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, in some cases, they cause behavior that is not intuitive. 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/2003 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/2003 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.

See also

- "UPDATE statement" [*SQL Anywhere Server - SQL Reference*]

ansinull option [compatibility]

Function

Controls the interpretation of NULL values.

Allowed values

On, Off

Default

On

Description

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.

auditing option [database]

Function

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 permissions are required to set this option.

Description

This option turns auditing on and off.

Auditing is the recording of detailed information 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.

See also

- "sa_enable_auditing_type system procedure" [*SQL Anywhere Server - SQL Reference*]
- "sa_disable_auditing_type system procedure" [*SQL Anywhere Server - SQL Reference*]

Example

Turn on auditing

```
SET OPTION PUBLIC.auditing = 'On';
```

auditing_options option [database]

This option is reserved for system use. Do not change the setting of this option.

automatic_timestamp option [compatibility]

Function

Controls the interpretation of new columns with the TIMESTAMP data type.

Allowed values

On, Off

Default

Off

Description

Controls whether any new columns with the TIMESTAMP data type that do not have an explicit default value defined are given a default value of the Transact-SQL timestamp value as a default. The automatic_timestamp option is included for Transact-SQL compatibility.

☞ For more information, see "Setting options for Transact-SQL compatibility" [*SQL Anywhere Server - SQL Usage*].

background_priority option [database]

Function

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.

Description

When set to On, it requests that the current connection have minimal impact on the performance of other connections. This option allows tasks for which responsiveness is critical to coexist with other tasks for which performance is not as important.

blob_threshold option [SQL Remote]

Function

Controls the size of value that the Message Agent treats as a long object (BLOB).

Allowed values

Integer, in kilobytes

Default

256

Description

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*]

blocking option [database]

Function

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.

Description

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.

See also

- ♦ “blocking_timeout option [database]” on page 398

blocking_timeout option [database]

Function

To control 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.

Description

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 [database]" on page 398

chained option [compatibility]

Function

Controls the transaction mode in the absence of a BEGIN TRANSACTION statement.

Allowed values

On, Off

Default

On

Off for Open Client and jConnect connections

Description

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.

checkpoint_time option [database]

Function

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 is required to set the option. You must shut down and restart the database server for the change to take effect.

Description

This option is used with the recovery_time option to decide when checkpoints should be done.

☞ For information on checkpoints, see "Checkpoints and the checkpoint log" on page 783.

See also

♦ "recovery_time option [database]" on page 449

cis_option option [database]

Function

Controls whether debugging information for remote data access appears in the Server Messages window.

Allowed values

0, 7

Default

0

Scope

Can be set for an individual connection or for the PUBLIC group.

Description

This option controls whether information about how queries are executed on a remote database appears in the Server Messages window when using remote data access. Set this option to 7 to see debugging information in the Server Messages window. When this option is set to 0 (the default), debugging information for remote data access does not appear in the Server Messages window.

Once you have turned on remote tracing, the tracing information appears in the 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 server option" on page 161.

cis_rowset_size option [database]

Function

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.

Description

This option sets the ODBC FetchArraySize value when using ODBC to connect to a remote database server.

close_on_endtrans option [compatibility]

Function

Controls the closing of cursors at the end of a transaction.

Allowed values

On, Off

Default

On

Off for jConnect connections

Description

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 [compatibility]” on page 391

collect_statistics_on_dml_updates option [database]

Function

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

Description

The server updates statistics during normal statement execution and uses the gathered statistics to self-tune the 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 accordingly. 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 [database]” on page 468
- “Updating column statistics” [*SQL Anywhere Server - SQL Usage*]

compression option [SQL Remote]

Function

Sets the level of compression for SQL Remote messages.

Allowed values

Integer, from -1 to 9

Default

6

Description

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.

conn_auditing option [database]

Function

Controls whether auditing is enabled or disabled for each connection when the auditing option is set to On.

Allowed values

On, Off

Default

On

Scope

Can be set as a temporary option only, for the duration of the current connection. DBA permissions are required to set this option.

Description

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

- "Turning on auditing" on page 868
- "auditing option [database]" on page 395
- "login_procedure option [database]" on page 424

continue_after_raiserror option [compatibility]

Function

Controls behavior following a RAISERROR statement.

Allowed values

On, Off

Default

On

Description

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 terminates.

Intermediate RAISERROR statuses and codes are lost after the procedure terminates. 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 [compatibility]" on page 438

conversion_error option [compatibility]

Function

Controls the reporting of data type conversion failures on fetching information from the database.

Allowed values

On, Off

Default

On

Description

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.

cooperative_commit_timeout option [database]

Function

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.

Description

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 [database]" on page 405

cooperative_commits option [database]

Function

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.

Description

If cooperative_commits is set to Off, a COMMIT is written to disk as soon as 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 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 [database]” on page 411

date_format option [compatibility]

Function

Sets the format for dates retrieved from the database.

☞ For information about controlling the interpretation of date formats, see “date_order option [compatibility]” on page 407.

Allowed values

String

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Default

'YYYY-MM-DD' (this corresponds to ISO date format specifications)

Description

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
d	Single digit day of week, (0 = Sunday, 6 = 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.

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.

See also

- “time_format option [compatibility]” on page 462
- “timestamp_format option [compatibility]” on page 463

Example

The following table illustrates date_format settings, together with the output from the following statement, executed on Friday June 2, 2006.

```
SELECT CURRENT DATE
```

date_format	SELECT CURRENT DATE
yyyy/mm/dd/ddd	2006/06/02/fri
yyyy/Mm/Dd/ddd	2006/6/2/fri
jjj	153
mmm yyyy	jun 2006
Mmm yyyy	Jun 2006
mm-yyyy	06-2006

date_order option [compatibility]

Function

Controls the interpretation of date formats.

☞ For information about setting the format for dates retrieved from the database, see "date_format option [compatibility]" on page 406.

Allowed values

MDY, YMD, DMY

Default

YMD (this corresponds to ISO date format specifications)

For 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.

Description

The database option date_order is used to determine whether 10/11/12 is Oct 11 1912, Nov 12 1910, or Nov 10 1912.

debug_messages option [database]

Function

Controls whether or not MESSAGE statements that include a DEBUG ONLY clause are executed.

Allowed values

On, Off

Default

Off

Description

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*]

dedicated_task option [database]

Function

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 permissions are required to set this option.

Description

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.

default_dbspace option [database]

Function

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.

Description

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.

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.

See also

- “Place different files on different devices” [*SQL Anywhere Server - SQL Usage*]
- “CREATE DBSPACE statement” [*SQL Anywhere Server - SQL Reference*]

Example

In the following example, a new dbspace named library is created. The default dbspace is then set to the library dbspace and the table LibraryBooks is stored in the library dbspace instead of the system dbspace.

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

default_timestamp_increment option [database] [MobiLink client]

Function

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 60000000 inclusive

Default

1

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Description

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 prior to performing the first synchronization.

See also

- “truncate_timestamp_values option [database] [MobiLink client]” on page 464

delayed_commit_timeout option [database]

Function

Determines when the server returns control to an application following a COMMIT.

Allowed values

Integer, in milliseconds

Default

500

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Description

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.

☞ For more information, see "delayed_commits option [database]" on page 411.

delayed_commits option [database]

Function

Determines when the server returns control to an application following a COMMIT.

Allowed values

On, Off

Default

Off (this corresponds to ISO COMMIT behavior)

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Description

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 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 cost of security.

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

- “cooperative_commit_timeout option [database]” on page 404
- “cooperative_commits option [database]” on page 405

delete_old_logs option [MobiLink client] [SQL Remote] [Replication Agent]

Function

Controls whether transaction logs are deleted when their transactions have been replicated or synchronized.

Allowed values

On, Off, Delay, *n* days

Default

Off

Description

This option is used by SQL Anywhere MobiLink clients, by SQL Remote, and by the SQL Anywhere Replication Agent. 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 older than the specified number of days 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 the “BACKUP statement” [*SQL Anywhere Server - SQL Reference*].

Example

To delete old logs that were created more than ten days ago:

```
delete_old_logs = 10 days
```

divide_by_zero_error option [compatibility]

Function

Controls the reporting of division by zero.

Allowed values

On, Off

Default

On

Description

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.

escape_character option [compatibility]

This option is reserved for system use. Do not change the setting of this option.

exclude_operators option [database]

This option is reserved for system use. Do not change the setting of this option.

extended_join_syntax option [database]

Function

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

Description

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*]

fire_triggers option [compatibility]

Function

Controls whether triggers are fired in the database.

Allowed values

On, Off

Default

On

Description

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.

first_day_of_week option [database]

Function

Sets the numbering of the days of the week.

Allowed values

1, 2, 3, 4, 5, 6, 7

Default

7 (Sunday is the first day of the week)

Description

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*]
- ♦ "SET statement [T-SQL]" [*SQL Anywhere Server - SQL Reference*]

float_as_double option [compatibility]

Function

Controls the interpretation of the FLOAT keyword.

Allowed values

On, Off

Default

Off

On for Open Client and jConnect connections

Description

The float_as_double option makes the FLOAT keyword behave like Adaptive Server Enterprise's FLOAT keyword when a precision is not specified.

When enabled (set to On), all occurrences of the keyword FLOAT are interpreted as equivalent to the keyword DOUBLE within SQL statements.

By default, SQL Anywhere FLOAT values are interpreted by Adaptive Server Enterprise as REAL values. Since Adaptive Server Enterprise treats its own FLOAT values as DOUBLE, enabling this option makes SQL Anywhere treat FLOAT values in the same way Enterprise treats FLOAT values.

REAL values are four bytes; DOUBLE values are eight bytes. According to the ANSI SQL/2003 specification, FLOAT can be interpreted based on the platform. It is up to the database to decide what size the value is, so long as it can handle the necessary precision. Adaptive Server Enterprise and SQL Anywhere exhibit different default behavior.

The float_as_double option takes effect only when no precision is specified. For example, the following statement is not affected by the option setting:

```
CREATE TABLE t1(
   c1 float( 5 ) );
```

The following statement is affected by the option setting:

```
CREATE TABLE t2(
   c1 float);
// affected by option setting
```

for_xml_null_treatment option [database]

Function

Controls the treatment of NULL values in queries that use the FOR XML clause.

Allowed values

Empty, Omit

Default

Omit

Description

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*]

force_view_creation option [database]

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 [database]

Function

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

Integer

Default

2147483647

Scope

Can be set for the PUBLIC group only. DBA authority required.

Description

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 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 GLOBAL DEFAULT AUTOINCREMENT is disabled. In this case NULL is generated as a default.

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*]
- "Database-level properties" on page 506
- MobiLink: "Setting the global database ID" [*MobiLink - Server Administration*]
- SQL Remote: "Ensuring unique primary keys" [*SQL Remote*]

Example

The following example sets the database identification number to 100.

```
SET OPTION PUBLIC.global_database_id = '100';
```

http_session_timeout option [database]

Function

Specify the amount of time, in minutes, that the client waits for an HTTP session to time out before giving up.

Allowed values

Integer (1 to 525600)

Default

30

Scope

Can be set for the PUBLIC group only. DBA authority is required to set this option.

Description

This option provides variable session timeout control for web service applications. A web service application can change the timeout value from within any request that owns the HTTP session, but a change to the timeout value can impact subsequent queued requests if the HTTP session times out. The web application must include logic to detect whether a client is attempting to access an HTTP session that no longer exists. This can be done by examining the value of the SessionCreateTime connection property to determine whether a timestamp is valid: if the HTTP request is not associated with the current HTTP session, then the SessionCreateTime connection property contains an empty string.

See also

- SessionCreateTime property and http_session_timeout property in "Connection-level properties" on page 476
- "Using HTTP sessions" [*SQL Anywhere Server - Programming*]

integrated_server_name option [database]

Function

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 is required to set this option.

Description

This option allows the DBA 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 84

- ♦ "GRANT statement" [*SQL Anywhere Server - SQL Reference*]

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 [compatibility]

Function

Controls the locking isolation level.

Allowed values

0, 1, 2, 3, snapshot, statement-snapshot, readonly-statement-snapshot

Default

0

1 for Open Client, jConnect, and TDS connections

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Description

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 isolation level specified by the updatable_statement_isolation option (can be one of 0 (the default), 1, 2, or 3).

☞ For more information, 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 iAnywhere 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.

See also

- "allow_snapshot_isolation option [database]" on page 389
- "updatable_statement_isolation option [database]" on page 467
- "Snapshot isolation" [*SQL Anywhere Server - SQL Usage*]
- "Isolation levels and consistency" [*SQL Anywhere Server - SQL Usage*]
- "Choosing isolation levels" [*SQL Anywhere Server - SQL Usage*]

java_location option [database]

Function

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 is required to set this option.

Description

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_main_userid option [database]" on page 420
- "java_vm_options option [database]" on page 421
- JavaVM property: "Database-level properties" on page 506
- "Choosing a Java VM" [*SQL Anywhere Server - Programming*]

java_main_userid option [database]

Function

Specifies the database user whose connection can be used for installing classes and other Java-related administrative tasks.

Allowed values

String

Default

DBA user (the default user created when the database is initialized)

Scope

Can be set for the PUBLIC group only. DBA authority required.

Description

The specified user ID should have DBA authority so they can perform the required operations. The password for this user is not required.

See also

- "java_location option [database]" on page 420
- "java_vm_options option [database]" on page 421
- "Choosing a Java VM" [*SQL Anywhere Server - Programming*]

java_vm_options option [database]

Function

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.

Description

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. In some cases, additional options are required to use the Java VM in 64-bit mode instead of 32-bit mode.

See also

- "java_location option [database]" on page 420
- "java_main_userid option [database]" on page 420
- "Choosing a Java VM" [*SQL Anywhere Server - Programming*]

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'
```

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 [database]

Function

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.

Description

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 cleared 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.

See also

- ♦ "sa_report_deadlocks system procedure" [*SQL Anywhere Server - SQL Reference*]
- ♦ "Determining who is blocked" [*SQL Anywhere Server - SQL Usage*]

login_mode option [database]

Function

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.

Description

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 whitespace. 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 integrated logins" on page 84
- "Using Kerberos authentication" on page 93
- "Security concerns: Copied database files" on page 102

Examples

Enable only integrated logins (standard logins and Kerberos logins fail):

```
SET OPTION PUBLIC.login_mode = 'Integrated';
```

Enable standard and Kerberos logins (integrated logins fail):

```
SET OPTION PUBLIC.login_mode = 'Standard,Kerberos';
```

Enable standard, integrated, and Kerberos logins:

```
SET OPTION PUBLIC.login_mode = 'Standard,Integrated,Kerberos';
```

login_procedure option [database]

Function

Specifies a login procedure that sets connection compatibility options at startup. By default the procedure calls the sp_login_environment procedure to determine which options to set.

Allowed values

String

Default

sp_login_environment

Scope

DBA authority required.

Description

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.

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.

See also

- ♦ "post_login_procedure [database]" on page 443
- ♦ "Increasing password security" on page 861

Examples

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 an alternate way to disallow connections, see "RAISERROR statement [T-SQL]" [*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
        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
```

materialized_view_optimization option [database]

Function

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.

Description

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, as well as 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.

You can override the setting of the materialized_view_optimization option for individual queries by specifying the acceptable level of staleness in the OPTION clause of the SELECT statement. See "SELECT statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "Refreshing materialized views" [*SQL Anywhere Server - SQL Usage*]
- "Working with materialized views" [*SQL Anywhere Server - SQL Usage*]
- "Enabling and disabling optimizer use of a materialized view" [*SQL Anywhere Server - SQL Usage*]

max_cursor_count option [database]

Function

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 permissions are required to set this option for *any* connection.

Description

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).

max_plans_cached option [database]

Function

Specifies the maximum number of execution plans to be stored in a cache.

Allowed values

Integer

Default

20

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately. DBA permissions are required to set this option for the PUBLIC group.

Description

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

- "Execution plan caching" [*SQL Anywhere Server - SQL Usage*]
- "Connection-level properties" on page 476

max_query_tasks option [database]

Function

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.

Description

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 22 and “Setting the database server's multiprogramming level” on page 25.

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.

☞ For more information about the exchange operator, see “Exchange algorithm” [*SQL Anywhere Server - SQL Usage*].

See also

- “-gn server option” on page 149
- “-gt server option” on page 152
- “-gtc server option” on page 153
- “Parallelism during query execution” [*SQL Anywhere Server - SQL Usage*]
- “Server-level properties” on page 496

max_recursive_iterations option [database]

Function

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 permissions are required to set this option for the PUBLIC group.

Description

Computation of a recursive common table expression aborts and an error is generated if the computation fails to terminate within the specified number of iterations. Recursive subqueries often increase geometrically in 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

- "Common Table Expressions" [*SQL Anywhere Server - SQL Usage*]

max_statement_count option [database]

Function

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 permissions are required to set this option for *any* connection.

Description

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 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

Interface	Statement
embedded SQL	DROP STATEMENT SQLFreeStmt(hstmt, SQL_DROP) or SQLFreeHandle(SQL_HANDLE_STMT, hstmt) resultSet.Close or Statement.Close RecordSet.Close SADataReader.Close or SaDataReader.Dispose
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.

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 server vulnerable to termination with an out-of-memory condition for applications that do not properly free prepared statements.

See also

- "Preparing statements" [*SQL Anywhere Server - Programming*]
- "DROP STATEMENT statement [ESQL]" [*SQL Anywhere Server - SQL Reference*]

max_temp_space option [database]

Function

Controls the maximum amount of temporary file space a connection can use.

Allowed values

Integer [**k** | **m** | **g** | **p**]

Default

0

Scope

Takes effect immediately. DBA permissions are required to set this option.

Description

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.

See also

- "temp_space_limit_check option [database]" on page 461
- "sa_disk_free_space system procedure" [*SQL Anywhere Server - SQL Reference*]

Examples

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 [database]

Function

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 permissions are required to set this option.

Description

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 [database]" on page 470

Example

Set the minimum length for new passwords to 6 bytes.

```
SET OPTION PUBLIC.min_password_length = 6;
```

nearest_century option [compatibility]

Function

Controls the interpretation of two-digit years in string-to-date conversions.

Allowed values

Integer, between 0 and 100 inclusive

Default

50

Description

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.

non_keywords option [compatibility]

Function

Turns off individual keywords, allowing their use as identifiers.

Allowed values

String

Default

Empty string

Description

This option turns off individual keywords or all keywords introduced since a specific release of the product. 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.

In addition to specifying individual keywords, you can turn off all keywords since a specified release using one of the following special values in the list of keywords:

```
keywords_4_0_d, keywords_4_0_c, keywords_4_0_b, keywords_4_0_a,
keywords_4_0, keywords_5_0_01, keywords_5_0
```

The following statement prevents TRUNCATE and SYNCHRONIZE from being recognized as keywords:

```
SET OPTION non_keywords = 'TRUNCATE, SYNCHRONIZE';
```

The following statement prevents all keywords introduced since version 4.0.d from being recognized as keywords:

```
SET OPTION non_keywords = 'keywords_4_0_d';
```

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.

See also

- "Keywords" [*SQL Anywhere Server - SQL Reference*]

odbc_describe_binary_as_varbinary [database]

Function

Controls how the SQL Anywhere ODBC driver describes BINARY columns.

Allowed values

On, Off

Default

Off

Description

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.

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_distinguish_char_and_varchar option [database]

Function

Controls how the SQL Anywhere ODBC driver describes CHAR columns.

Allowed values

On, Off

Default

Off

Description

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.

The odbc_distingish_char_and_varchar option also controls whether NCHAR columns are described as SQL_WCHAR or SQL_WVARCHAR. If this option is set to Off, then NCHAR columns are described as SQL_WVARCHAR. If this option is set to On, then NCHAR columns are described as SQL_WCHAR. NVARCHAR columns are always described as SQL_WVARCHAR.

See also

- "NCHAR data type" [*SQL Anywhere Server - SQL Reference*]
- "NVARCHAR data type" [*SQL Anywhere Server - SQL Reference*]

oem_string option [database]

Function

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 permissions are required to set this option.

Description

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 ISYSOPTIONS 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 Server Messages window.
- *dboem.pl* illustrates how to extract the OEM string and print it to the stdout within a PERL script.

Caution
Applications cannot write directly to the OEM string in the database because it will corrupt 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*]

Example

The following example encrypts the OEM string that contains information about the database file and stores it in the database header file:

```
SET OPTION PUBLIC.oem_string=
 BASE64_ENCODE(
    ENCRYPT( 'Database version 10', '4j67lm3z' ) );
```

You can retrieve the value of the OEM string using the following command:

```
SELECT DECRYPT(
   BASE64_DECODE(
   CONNECTION_PROPERTY( 'oem_string' ), '4j67lm3z' ) );
```

on_charset_conversion_failure option [database]

Function

Controls what happens if an error is encountered during character conversion.

Allowed values

Ignore, Warning, Error

Default

Ignore

Description

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.

Single-byte to single-byte converters are not able to report substitutions and illegal characters, and must be set to Ignore.

on_tsql_error option [compatibility]

Function

Controls error-handling in stored procedures.

Allowed values

String (see below for allowed values)

Default

Conditional

Continue for jConnect connections

Description

This option controls error handling in stored procedures.

- **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.

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. The Conditional setting is recommended, particularly when developing new Transact-SQL stored procedures, as it allows errors to be reported earlier.

When this option is set to Stop or Continue, it supercedes 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 [compatibility]" on page 403

optimistic_wait_for_commit option [compatibility]

Function

Controls locking behavior for the wait_for_commit option.

Allowed values

On, Off

Default

Off

Description

By default, optimistic_wait_for_commit is Off.

Locking behavior is changed as follows when both optimistic_wait_for_commit and wait_for_commit are On:

- No locks are placed on primary key rows when adding a foreign key row without a matching primary key row.
- If a transaction adds a primary row, it will be allowed to commit only if the transaction has exclusive locks on all foreign rows that reference the primary row at the time the primary row is added.

This feature is for use in migrating version 5.x.x applications to version 8.0.x or later by mimicking version 5.x.x locking behavior when transactions add foreign rows before primary rows (as long as no two transactions concurrently add foreign rows with the same key value).

Note that this option is not recommended for general use as there are a number of situations in which transactions would not be allowed to commit, including:

- If transactions concurrently add foreign rows with the same key value when the primary row does not exist, at most 1 of the transactions will be allowed to commit (the one that adds the corresponding primary row).
- If a transaction deletes a primary row and then adds it back, it likely would not be allowed to commit (unless it has somehow obtained exclusive locks on all of the matching foreign rows).

See also

- “wait_for_commit option [database]” on page 472

optimization_goal option [database]

Function

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

Description

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 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*].

optimization_level option [database]

Function

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

Description

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, and it is known that although the statement may be complex, the query's execution time is very small, and hence 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-level properties” on page 476.

optimization_workload option [database]

Function

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.

Description

The optimization_workload option controls whether SQL Anywhere optimizes queries for a workload that is a mix of updates and reads or 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.

See also

- "Clustered Hash Group By algorithm" [*SQL Anywhere Server - SQL Usage*]

percent_as_comment option [compatibility]

Function

Controls the interpretation of the percent character.

Allowed values

On, Off

Default

On

Description

It is recommended that you do not use % as a comment marker.

Versions of this product before Version 6 treated the percent character (%) in SQL statements exclusively as a comment delimiter. Since Version 5, alternative comment markers such as ***//***, ***/* */***, and **–** (double dash) have been available. The double-dash style is the SQL/2003 comment delimiter.

Adaptive Server Enterprise treats **%** as a modulo operator and does not support the SQL Anywhere `mod` function. Writing a statement that works in both environments and performs a modulo operation was previously impossible.

The percent_as_comment option controls the meaning of %. The default setting is On for backward compatibility. You can set the option to Off for compatibility with Adaptive Server Enterprise.

Procedures, triggers and views that were created with %-style comments are converted to double-dash comments when they are stored in the catalog.

pinned_cursor_percent_of_cache option [database]

Function

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.

Description

The 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.

post_login_procedure [database]

Function

Specifies a procedure whose result set contains messages that should be displayed by applications when a user connects.

Allowed values

String

Default

Empty string

Scope

DBA authority required.

Description

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 plug-in for Sybase Central, Interactive SQL, and dbisqlc call the procedure if this option is set and display any messages returned by the procedure in a dialog. 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 plug-in, dbisql, and dbisqlc display all non-NULL messages, regardless of the action value. If the action is set to 1, then the SQL Anywhere plug-in and dbisql (but not dbisqlc) 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 [database]" on page 424
- "Increasing password security" on page 861

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';
```

precision option [database]

Function

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 an individual connection or for the PUBLIC group. Takes effect immediately.

Description

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.

prefetch option [database]

Function

Controls whether or not 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.

Description

This option controls whether rows are fetched to the client side in advance of being made available to the client application. Fetching a number of 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 in some cases when GET DATA is used.

The setting of the prefetch option is ignored by 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 connection parameter [DMRF]” on page 221

preserve_source_format option [database]

Function

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

Scope

Can be set for the PUBLIC group only. DBA authority required.

Description

When preserve_source_format is On, the 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.

prevent_article_pkey_update option [database] [MobiLink client]

Function

Controls updates to the primary key columns of tables involved in publications.

Allowed values

On, Off

Default

On

Description

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.

qualify_owners option [SQL Remote]

Function

Controls whether SQL statements being replicated by SQL Remote should use qualified object names.

Allowed values

On, Off

Default

On

Description

When qualification is not needed in SQL Anywhere setups, messages will be slightly smaller with this option set to Off.

query_plan_on_open option [compatibility]

Function

Controls whether a plan is returned when a cursor is opened.

Allowed values

On, Off

Default

Off

Description

In early versions of the software, each time an OPEN was done on a cursor, the server would return in the SQLCA **sqlerrmc** field a string representing the query plan (limited to 70 bytes). A more complete description can be obtained using the EXPLAIN statement or the PLAN function. For this reason, computing and returning the query plan on an OPEN is needed only for compatibility with old applications. The query_plan_on_open option controls whether the plan is returned on an OPEN.

quote_all_identifiers option [SQL Remote]

Function

Controls whether SQL statements being replicated by SQL Remote should use quoted identifiers.

Allowed values

On, Off

Default

Off

Description

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.

quoted_identifier option [compatibility]

Function

Controls the interpretation of strings that are enclosed in double quotes.

Allowed values

On, Off

Default

On

Off for Open Client and jConnect connections

Description

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.

☞ For more information, see “Setting options for Transact-SQL compatibility” [*SQL Anywhere Server - SQL Usage*].

read_past_deleted option [database]

Function

Controls server behavior on uncommitted deletes at isolation levels 1 and 2.

Allowed values

On, Off

Default

On

Description

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).

recovery_time option [database]

Function

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.

Description

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. Thus, the recovery time is not exact.

☞ For more information, see "The automatic recovery process" on page 785.

See also

- "checkpoint_time option [database]" on page 399

remote_idle_timeout option [database]

Function

Controls how many seconds of inactivity web service client procedures and functions will tolerate.

Allowed values

Integer, in seconds

Default

15

Description

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.

replicate_all option [Replication Agent]

Function

Allows an entire database to act as a primary site in a Replication Server setup.

Allowed values

On, Off

Default

Off

Description

This option is only used by the SQL Anywhere Replication Agent. When it is set to On, the entire database is set to act as a primary site in a Replication Server installation. All changes to the database are sent to Replication Server by the Replication Agent.

See also

- "Replicating an entire database" on page 956

replication_error option [SQL Remote]

Function

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

Description

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.

replication_error_piece option [SQL Remote]

Function

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

Description

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 451

request_timeout option [database]

Function

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.

Allowed values

Integer, 0 through 86400 (one day), in seconds

Default

0

Description

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").

User and public values 1 to 14 are not allowed. This prevents users from being locked out of the 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 [database]” on page 398
- “AppInfo connection parameter [APP]” on page 207

return_date_time_as_string option [database]

Function

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.

Description

This option indicates whether date, time, and timestamp values are returned to applications as a date or time datatype or as a string.

When this option is set to On, the 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 [compatibility]" on page 406
- "time_format option [compatibility]" on page 462
- "timestamp_format option [compatibility]" on page 463

ri_trigger_time option [compatibility]

Function

Controls the relative timing of referential integrity checks and trigger actions.

Allowed values

Before, After

Default

After

Scope

Can be set for the PUBLIC group only. DBA authority required.

Description

The option can be set to either Before or After. When it is set to After, referential integrity actions are executed after the UPDATE or DELETE.

Only the PUBLIC setting can be used; any other setting is ignored.

rollback_on_deadlock [database]

Function

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, as well as individual connections. Takes effect immediately.

Description

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.

☞ For more information about deadlocks, see "Deadlock" [*SQL Anywhere Server - SQL Usage*].

row_counts option [database]

Function

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.

Description

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.

Warning
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.

scale option [database]

Function

Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

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 an individual connection or for the PUBLIC group. Takes effect immediately.

Description

Multiplication, division, addition, subtraction, and aggregate functions can all have results that exceed the maximum precision.

☞ For an example, see "precision option [database]" on page 444.

secure_feature_key [database]

Function

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.

Description

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 server option" on page 173
- "-sf server option" on page 170
- "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*]
- "Specifying secured features" on page 866

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.

```
dbsrv10 -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 [database]

Function

Allows implicit use of the SORTKEY function on ORDER BY expressions.

Allowed values

Internal, collation_name, or collation_id

Default

Internal

Description

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, any string expression in the ORDER BY clause is treated as if the SORTKEY function had been invoked.

See also

- "SORTKEY function [String]" [*SQL Anywhere Server - SQL Reference*]

Example

Set the sort collation to binary:

```
SET TEMPORARY OPTION sort_collation='binary';
```

Having the sort collation set to binary transforms the following queries:

```
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 [compatibility]

Function

Controls the response to any SQL that is not part of a specified set of SQL/2003.

Allowed values

E, I, F, W

Default

W

Description

This option flags as an error any SQL that is not part of the specified standard.

The allowed values are as follows:

- **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

sql_flagger_warning_level option [compatibility]

Function

Controls the response to any SQL that is not part of a specified set of SQL/2003.

Allowed values

E, I, F, W

Default

W

Description

This option flags any SQL that is not part of a specified set of SQL/2003 as a warning.

The allowed values are as follows:

- **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

string_rtruncation option [compatibility]

Function

Determines whether an error is raised when an INSERT or UPDATE truncates a CHAR or VARCHAR string.

Allowed values

On, Off

Default

On

Description

If the truncated characters consist only of spaces, no exception is raised. The setting of On corresponds to ANSI/ISO SQL/2003 behavior. When this option is set to Off, the exception is not raised and the character string is silently truncated.

See also

- "Character data types" [*SQL Anywhere Server - SQL Reference*]

subscribe_by_remote option [SQL Remote]

Function

Controls interpretation of NULL or empty-string SUBSCRIBE BY values.

Allowed values

On, Off

Default

On

Description

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.

subsume_row_locks option [database]

Function

Controls when the server acquires individual row locks for a table.

Allowed values

On, Off

Default

On

Description

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 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 server could turn an updatable cursor with an ORDER BY into a keyset cursor as a result.

suppress_tds_debugging option [database]

Function

Determines whether TDS debugging information appears in the Server Messages window.

Allowed values

On, Off

Default

Off

Description

When the server is started with the -z option, debugging information appears in the Server Messages window, including debugging information about the TDS protocol.

The suppress_tds_debugging option restricts the debugging information about TDS that appears in the Server Messages window. When this option is set to Off (the default) TDS debugging information appears in the Server Messages window.

synchronize_mirror_on_commit option [database]

Function

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

Description

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 acknowledgment 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

♦ “Understanding database mirroring” on page 828

tds_empty_string_is_null option [database]

Function

Controls whether empty strings are returned as NULL or a string containing one blank character for TDS connections.

Allowed values

On, Off

Default

Off

Description

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.

temp_space_limit_check option [database]

Function

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.

Description

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.

See also

- "sa_disk_free_space system procedure" [*SQL Anywhere Server - SQL Reference*]
- "max_temp_space option [database]" on page 431

time_format option [compatibility]

Function

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.

Description

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 [compatibility]" on page 406
- "timestamp_format option [compatibility]" on page 463

time_zone_adjustment option [database]

Function

Allows a connection's time zone adjustment to be modified.

Allowed values

Integer (for example, 300)

Negative integer enclosed in quotation marks (for example, '-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 Open Client, the default is based on the server's time zone.

Description

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.

See also

- "Connection-level properties" on page 476

timestamp_format option [compatibility]

Function

Sets the format for timestamps that are retrieved from the database.

Allowed values

String (composed of the symbols listed below)

Default

YYYY-MM-DD HH:NN:SS.SSS

For Open Client and JDBC connections the default is set to *YYYY-MM-DD HH:NN:SS.SSS*

Scope

Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Description

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

Symbol	Description
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.

See also

- “date_format option [compatibility]” on page 406
- “time_format option [compatibility]” on page 462

truncate_timestamp_values option [database] [MobiLink client]

Function

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.

Description

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 prior to 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.

In most cases, 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 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 server when the row is 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 [database] [MobiLink client]” on page 410

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.

truncate_with_auto_commit option [database]

Function

Speeds up TRUNCATE TABLE statements.

Allowed values

On, Off

Default

On

Scope

Can be set for the PUBLIC group only. DBA authority required.

Description

If truncate_with_auto_commit is set to On, then a COMMIT is executed both before and after the TRUNCATE TABLE statement is executed. The primary purpose of the option is to enable faster table truncation (delete of all rows).

There are some cases where a fast TRUNCATE cannot be done:

- If there are foreign keys either to or from the table
- If the TRUNCATE TABLE statement is executed within a trigger
- If the TRUNCATE TABLE statement is executed within an atomic statement

See also

- "TRUNCATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]

tsql_hex_constant option [compatibility]

Function

Controls whether hexadecimal constants are treated as binary typed constants.

Allowed values

On, Off

Default

On

Description

When this option is set to On, hexadecimal constants are treated as binary typed constants. To get the historical behavior, set the option to Off.

tsql_outer_joins option [compatibility]

Function

Controls the ability to use the TSQL outer join operators *= and =* in statements and views.

Allowed values

On, Off

Default

Off

Description

Support for TSQL outer joins is deprecated as of version 10 and will be removed in a future release. This option enables the ability to use TSQL outer joins.

tsql_variables option [compatibility]

Function

Controls whether the @ sign can be used as a prefix for embedded SQL host variable names.

Allowed values

On, Off

Default

Off

On for Open Client and jConnect connections

Description

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.

updatable_statement_isolation option [database]

Function

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

Description

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.

See also

- "isolation_level option [compatibility]" on page 419
- "Snapshot isolation" [*SQL Anywhere Server - SQL Usage*]
- "Isolation levels and consistency" [*SQL Anywhere Server - SQL Usage*]
- "Choosing isolation levels" [*SQL Anywhere Server - SQL Usage*]

update_statistics option [database]

Function

Controls the gathering of statistics during query execution.

Allowed values

On, Off

Default

On

Description

The server collects statistics during normal query execution and uses the gathered statistics to self-tune the column statistics. You can set the update_statistics option Off to disable the gathering of statistics during query execution.

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 accordingly. 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 [database]" on page 401

- “Updating column statistics” [*SQL Anywhere Server - SQL Usage*]

user_estimates option [database]

Function

Controls whether or not 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.

Description

SQL Anywhere allows you to specify user selectivity estimates can improve the optimizer's performance when the 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 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.

uuid_has_hyphens option [database]

Function

Controls the formatting of unique identifier values when they are converted to strings.

Allowed values

On, Off

Default

On

Description

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]

Function

Controls whether messages that contain updates published by the local database are sent with all column values included.

Allowed values

On, Off

Default

Off

Description

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.

verify_password_function option [database]

Function

Specifies a function that can be used to implement password rules (for example, passwords must include at least one digit). The function is called on a GRANT CONNECT TO *userid* IDENTIFIED BY *password* statement.

Allowed values

String

Default

Empty string (no function is called on a GRANT CONNECT statement)

Scope

DBA authority required.

Description

When the verify_password_function option value is set to anything other than an empty string, a GRANT CONNECT TO *userid* IDENTIFIED BY *password* statement calls the function specified by the option value. The option value should be of the form *owner.function_name* to prevent a user from overriding the function. Note that SQL Anywhere passwords are case sensitive.

Once it has been determined that the GRANT CONNECT statement is valid (for example, the user has permission to perform the grant), then the function specified by this option is called to verify the password by the rules it specifies. If the password conforms to the specified rules, the function must return NULL to indicate success, and the grant is performed. 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 the GRANT CONNECT statement is not allowed.

☞ For an example that includes advanced password rules such as disallowing password reuse and implementing password expiration, see “Use a password verification function” on page 861.

See also

- “min_password_length option [database]” on page 432
- “GRANT statement” [*SQL Anywhere Server - SQL Reference*]
- “ALTER FUNCTION statement” [*SQL Anywhere Server - SQL Reference*]

Example

The following example creates a function named f_verify_pwd that checks whether the password is the same as a user's user ID. If the password and user ID are the same, then the user must specify a different password.

```
CREATE FUNCTION DBA.f_verify_pwd( user_name VARCHAR(128),
                                  new_pwd VARCHAR(255) )
RETURNS VARCHAR(255)
BEGIN
    -- enforce password rules
    IF new_pwd = user_name THEN
        RETURN( 'password cannot be the same as the user name' );
    END IF;
    -- return success
    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';
```

verify_threshold option [SQL Remote]

Function

Controls which columns are verified when updates are replicated.

Allowed values

Integer, in bytes

Default

1000

Description

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.

wait_for_commit option [database]

Function

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.

Description

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.

webservice_namespace_host option [database]

Function

Specifies the hostname to be used as the XML namespace within generated WSDL documents. Requires DBA authority.

Allowed values

NULL or hostname-string

Scope

Can be set for the PUBLIC group only. Takes effect immediately. DBA authority is required to set this option.

Default

NULL

Description

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.

CHAPTER 13

Database Properties

Contents

About this chapter

This chapter contains information and tables relating to SQL Anywhere database, server, and connection properties.

Understanding database properties

SQL Anywhere provides a set of properties that are made available to client applications. These properties describe aspects of connection, database, and database server behavior. Property names are case insensitive.

Accessing properties

Each type of property can be accessed by supplying its name as an argument to a system function.

♦ To access connection properties

- 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 access database properties

- Use the DB_PROPERTY system function. For example, the following statement returns the page size of the current database:

```
SELECT DB_PROPERTY ( 'PageSize' );
```

♦ To access database server properties

- Use the PROPERTY system function: the following statement returns the number of cache pages used for global server data structures.

```
SELECT PROPERTY ( 'MainHeapPages' );
```

Connection-level properties

The following table lists properties available for each connection.

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

- “Server-level properties” on page 496
- “Database-level properties” on page 506

Descriptions

Property	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 [compatibility]” on page 389.
allow_snapshot_isolation	Returns a value indicating whether snapshot isolation is enabled or disabled. See “allow_snapshot_isolation option [database]” on page 389.
ansi_blanks	Returns a value indicating when character data is truncated at the client side. See “ansi_blanks option [compatibility]” on page 390.
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 [compatibility]” on page 391.
ansi_integer_overflow	Returns a value indicating what happens when an arithmetic expression causes an integer overflow error. See “ansi_integer_overflow option [compatibility]” on page 391.
ansi_permissions	Returns a value indicating whether permissions are checked for DELETE and UPDATE statements. See “ansi_permissions option [compatibility]” on page 392.
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 [compatibility]” on page 392.
ansi_update_constraints	Returns a value indicating the range of updates that are permitted. See “ansi_update_constraints option [compatibility]” on page 394.
ansinull	Returns a value that indicates how NULL values are interpreted. See “ansinull option [compatibility]” on page 394.

Property	Description
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 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 connection parameter [APP]" on page 207.
ApproximateCPUTime	Returns an estimate of the amount of CPU time accumulated by a given connection. The value returned may differ from the actual value by as much as 50%, although typical variations are in the 5-10% range. On multi-processor 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.
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 "Turning on auditing" on page 868 and "auditing option [database]" on page 395.
auditing_options	This property is reserved for system use. Do not change the setting of this option.
automatic_timestamp	Returns a value indicating how new columns with the TIMESTAMP data type are interpreted. See "automatic_timestamp option [compatibility]" on page 396.
background_priority	Returns a value indicating how much impact the current connection has on the performance of other connections. See "background_priority option [database]" on page 397.
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 locking conflicts. See"blocking option [database]" on page 398.
blocking_timeout	Returns the length of time, in milliseconds, a transaction waits to obtain a lock. See "blocking_timeout option [database]" on page 398.
BytesReceived	Returns the number of bytes received during client/server communications.

Property	Description
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.
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.
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 [compatibility]" on page 399.
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 [database]" on page 399.
cis_option	Returns 0 if debugging information for remote data access appears in the Server Messages window and 7 if the debugging information for remote data access does not appear in the Server Messages window. See "cis_option option [database]" on page 400.
cis_rowset_size	Returns the number of rows that are returned from remote servers for each fetch. See "cis_rowset_size option [database]" on page 400.

Property	**Description**
ClientLibrary	Returns jConnect for jConnect connections; CT_Library for Open Client connections; None for HTTP connections, and CmdSeq for ODBC, embedded SQL, OLE DB, ADO.NET, and iAnywhere JDBC driver connections.
ClientPort	Returns the client's TCP/IP port number or 0 if the connection isn't a TCP/IP connection.
close_on_endtrans	Returns On or Off to indicate whether cursors are closed at the end of a transaction. See "close_on_endtrans option [compatibility]" on page 401.
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 [database]" on page 401.
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.
CommNetworkLink	Returns the communication link for the connection. This is one of the network protocols supported by SQL Anywhere. Values include SharedMemory, TCPIP, and SPX. The CommNetworkLink property always returns the name of the link, regardless of whether it is same-computer or not.
CommProtocol	Returns TDS for Open Client and jConnect connections, HTTP for HTTP connections, and CmdSeq for ODBC, embedded SQL, OLE DB, ADO.NET, and iAnywhere JDBC driver connections.
Compression	Returns On or Off to indicate whether communication compression is enabled on the connection.
conn_auditing	Returns On if auditing is enabled for the connection, even if the auditing option is set to Off. See "Turning on auditing" on page 868.
connection_authentication	Returns the string used to authenticate the client. Authentication is required before the database can be modified.
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 [compatibility]" on page 403.
conversion_error	Returns On or Off to indicate data type conversion failures are reported when fetching information from the database. See "conversion_error option [compatibility]" on page 404.

Property	Description
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 [database]" on page 404.
cooperative_commits	Returns On or Off to indicate when commits are written to disk. See "cooperative_commits option [database]" on page 405.
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.
database_authentication	Returns the string used to authenticate the database. Authentication is required for authenticated database servers before the database can be modified.
date_format	Returns a string indicating the format for dates retrieved from the database. See "date_format option [compatibility]" on page 406.
date_order	Returns a string indicating how dates are formatted. See "date_order option [compatibility]" on page 407.
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 [database]" on page 408.
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 [database]" on page 409.
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 [database]" on page 409.

Property	Description
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 [database] [MobiLink client]" on page 410.
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 [database]" on page 410.
delayed_commits	Returns On or Off to indicate when the database server returns control to an application following a COMMIT. See "delayed_commits option [database]" on page 411.
DiskRead	Returns the number of pages that have been read from disk.
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.
DiskWrite	Returns the number of modified pages that have been written to disk.
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 [compatibility]" on page 412.
Encryption	Returns a value that indicates whether the connection is encrypted. See "Encryption connection parameter [ENC]" on page 222.
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.

Property	Description
ExprCacheInserts	Returns the number of values inserted into the expression cache.
ExprCacheLookups	Returns the number of lookups done in the expression cache.
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 a value indicating whether queries with duplicate correlation name syntax for multi-table joins are allowed, or reported as an error. See "extended_join_syntax option [database]" on page 413.
fire_triggers	Returns On if triggers are fired in the database, otherwise, returns Off. See "fire_triggers option [compatibility]" on page 414.
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 [database]" on page 414.
float_as_double	Returns On if occurrences of the FLOAT keyword are interpreted as equivalent to the keyword DOUBLE within SQL statements; otherwise, returns Off. See "float_as_double option [compatibility]" on page 415.
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 [database]" on page 416.
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 [database]" on page 417.
HashForcedPartitions	Returns the number of times that a hash operator was forced to partition because of competition for memory.
HashRowsFiltered	Returns the rate at which probe rows are rejected by bit-vector filters.
HashRowsPartitioned	Returns the rate at which rows are written to hash work tables.

Property	Description
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).
HeapsRelocatable	Returns the number of relocatable heaps.
http_session_timeout	Returns the current HTTPS session timeout, in minutes. See "http_session_timeout option [database]" on page 417.
HttpServiceName	Returns the service name origin for a web application. 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 227.
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 [database]" on page 418.
isolation_level	Returns the isolation level of the connection. See "isolation_level option [compatibility]" on page 419.
java_location	Returns the path of the Java VM for the database if one has been specified.
java_main_userid	Returns the name of the database user whose connection can be used for installing classes and other Java-related administrative tasks.
Language	Returns the locale language.
LastIdle	Returns the number of ticks between requests.

Property	Description
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 server option" on page 190.
LastReqTime	Returns the time at which the last request for the specified connection started.
LastStatement	Returns the most recently prepared SQL statement for the current connection. See "-zl server option" on page 187. 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.
LivenessTimeout	Returns the liveness timeout period for the current connection. See "LivenessTimeout connection parameter [LTO]" on page 231.
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.
LockName	Returns a 64-bit unsigned integer value representing the lock for which a connection is waiting.
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 [database]" on page 422.
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 [database]" on page 422.

Property	Description
login_procedure	Returns the name of the stored procedure used to set compatibility options at startup. See "login_procedure option [database]" on page 424.
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. See "materialized_view_optimization option [database]" on page 426.
max_cursor_count	Returns a value specifying the maximum number of cursors that a connection can use at once. See "max_cursor_count option [database]" on page 427.
max_plans_cached	Returns a value specifying the maximum number of execution plans to be stored in a cache. See "max_plans_cached option [database]" on page 427.
max_query_tasks	Returns the maximum number of requests that the database server can use to process a query. See "max_query_tasks option [database]" on page 428.
max_recursive_iterations	Returns a value specifying the maximum number of iterations a recursive common table expression can make. See "max_recursive_iterations option [database]" on page 429.
max_statement_count	Returns a value specifying the maximum number of prepared statements that a connection can use simultaneously. See "max_statement_count option [database]" on page 430.
max_temp_space	Returns a value indicating the maximum amount of temporary file space available for a connection. See "max_temp_space option [database]" on page 431.
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 [database]" on page 432.
Name	Returns the name of the current connection.
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 [compatibility]" on page 433.

Property	Description
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 [compatibility]" on page 433.
Number	Returns the ID number of the connection.
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 [database]" on page 434.
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 [database]" on page 435.
oem_string	Returns the string stored in the header page of the database file. See "oem_string option [database]" on page 435.
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 [database]" on page 437.
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 [compatibility]" on page 438.
optimistic_wait_for_commit	Returns On or Off to indicate the locking behavior for the wait_for_commit option. See "optimistic_wait_for_commit option [compatibility]" on page 438.
optimization_goal	Returns one of First-row or All-rows to indicate how query processing is optimized. See "optimization_goal option [database]" on page 439.
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 [database]" on page 440.
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 [database]" on page 441.
PacketSize	Returns the packet size used by the connection, in bytes.

Property	Description
PacketsReceived	Returns the number of client/server communication packets received.
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.
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.)
percent_as_comment	Returns On if the percent sign (%) is used as a comment marker, otherwise, returns Off. See "percent_as_comment option [compatibility]" on page 442.
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 [database]" on page 442.
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 [database]" on page 443.
precision	Returns the decimal and numeric precision setting. See "precision option [database]" on page 444.
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 [database]" on page 445.
Prepares	Returns the number of statement preparations performed.
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 [database]" on page 446.
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 [database] [MobiLink client]" on page 447.

Property	Description
query_plan_on_open	Returns a value indicating whether a plan is returned when a cursor is opened. See "query_plan_on_open option [compatibility]" on page 447.
QueryBypassed	Returns the number of requests optimized by the optimizer bypass.
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.
QueryHeapPages	Returns the number of cache pages used for query processing (hash and sort operations).
Query JHToJNLOptUsed	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.
QueryOptimized	Returns the number of requests that have been fully optimized.
QueryReused	Returns the number of requests that have been reused from the plan cache.
QueryRowsBufferFetch	Returns the number of rows fetched using buffering.
QueryRowsMaterialized	Returns the rate at which 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 [compatibility]" on page 448.
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 [database]" on page 449.
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 [database]" on page 449.
RecursiveIterations	Returns the number of iterations for recursive unions.
RecursiveIterationsHash	Returns the number of times recursive hash join used a hash strategy.

Property	Description
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.
remote_idle_timeout	Returns the time, in seconds, of inactivity that web service client procedures and functions will tolerate. See "remote_idle_timeout option [database]" on page 450.
replicate_all	Returns On if the database is acting as a primary site in a Replication Server installation; otherwise, returns Off. See "replicate_all option [Replication Agent]" on page 450.
ReqCountActive	Returns the number of requests processed, or NULL if the RequestTiming server property is set to Off. See "-zt server option" on page 192.
ReqCountBlockContention	Returns the number of times the connection waited for atomic access, or NULL if the -zt option was not specified. See "-zt server option" on page 192.
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 server option" on page 192.
ReqCountBlockLock	Returns the number of times the connection waited for a lock, or NULL if the -zt option was not specified. See "-zt server option" on page 192.
ReqCountUnscheduled	Returns the number of times the connection waited for scheduling, or NULL if the -zt option was not specified. See "-zt server option" on page 192.

Property	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 a worker thread. ♦ **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), they values are reported as Executing. ☞ For more information, see “-zt server option” on page 192.
ReqTimeActive	Returns the amount of time spent processing requests, or NULL if the -zt option was not specified. See “-zt server option” on page 192.
ReqTimeBlockContention	Returns the amount of time spent waiting for atomic access, or NULL if the RequestTiming server property is set to Off. See “-zt server option” on page 192.
ReqTimeBlockIO	Returns the amount of time spent waiting for I/O to complete, or NULL if the -zt option was not specified. See “-zt server option” on page 192.
ReqTimeBlockLock	Returns the amount of time spent waiting for a lock, or NULL if the -zt option was not specified. See “-zt server option” on page 192.
ReqTimeUnscheduled	Returns the amount of unscheduled time, or NULL if the -zt option was not specified. See “-zt server option” on page 192.
ReqType	Returns the type of the last request.
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	Description
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 datatype. See "return_date_time_as_string option [database]" on page 452.
ri_trigger_time	Returns Before or After to indicate when referential integrity actions are executed after an UPDATE or DELETE. See "ri_trigger_time option [compatibility]" on page 453.
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 [database]" on page 453.
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 [database]" on page 454.
scale	Returns the decimal and numeric scale for the connection. See "scale option [database]" on page 455.
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.
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 one has been defined, otherwise, returns an empty string.
SessionLastTime	Returns the time of the last request for the HTTP session.
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 [database]" on page 456.
SortMergePasses	Returns the number of merge passes used during sorting.
SortRowsMaterialized	Returns the rate at which rows are 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.

Property	Description
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 [compatibility]” on page 457.
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 [compatibility]” on page 457.
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 [compatibility]” on page 458.
subsume_row_locks	Returns On if the database server acquires individual row locks for a table, otherwise, returns Off. See “subsume_row_locks option [database]” on page 459.
suppress_tds_debugging	Returns Off if TDS debugging information appears in the Server Messages window, and returns On if debugging information does not appear in the Server Messages window. See “suppress_tds_debugging option [database]” on page 459.
synchronize_mirror_on_commit	Returns On if the database mirror server is synchronized on commit, otherwise returns Off. See “synchronize_mirror_on_commit option [database]” on page 460.
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 [database]” on page 460.

Property	Description
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 [database]" on page 461.
TempTablePages	Returns the number of pages in the temporary file used for temporary tables.
time_format	Returns the string format used for times retrieved from the database. See "time_format option [compatibility]" on page 462.
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 [compatibility]" on page 463.
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 [database]" on page 462.
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 [database] [MobiLink client]" on page 464.
truncate_with_auto_commit	Returns On if a COMMIT is executed both before and after the TRUNCATE TABLE statement is executed, otherwise, returns Off. See "truncate_with_auto_commit option [database]" on page 466.
tsql_hex_constant	Returns On if hexadecimal constants are treated as binary typed constants, otherwise, returns Off. See "tsql_hex_constant option [compatibility]" on page 466.
tsql_outer_joins	Returns whether Transact-SQL outer joins can be used in DML statements. See "tsql_outer_joins option [compatibility]" on page 467.
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 [compatibility]" on page 467.
UncommitOp	Returns the number of uncommitted operations.

Property	Description
update_statistics	This property is reserved for system use. Do not change the setting of this option.
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 [database]" on page 469.
UserAppInfo	Returns the string specified by the AppInfo connection parameter in a connection string. ☞ For more information, see "AppInfo connection parameter [APP]" on page 207.
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 server option" on page 154.
uuid_has_hyphens	Returns On if UUIDs contain four hyphens, and Off if they contain no hyphens. See "uuid_has_hyphens option [database]" on page 469.
verify_password_function	Returns the name of the function used for password verification if one has been specified. See "verify_password_function option [database]" on page 470.
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 [database]" on page 472.

Property	Description
webservice_namespace_host	Returns the hostname to be used as the XML namespace within generated WSDL documents if one has been specified. See “webservice_namespace_host option [database]” on page 472.

Server-level properties

The following table lists properties that apply across the server as a whole.

Examples

♦ **To retrieve the value of a 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 server properties**

- Use the sa_eng_properties system procedure:

```
CALL sa_eng_properties;
```

See also

♦ “Connection-level properties” on page 476

♦ “Database-level properties” on page 506

Descriptions

Property	Description
ActiveReq	Returns the number of server threads that are currently handling a request.
AvailIO	Returns the current number of available I/O control blocks.
BuildChange	Reserved.
BuildClient	Reserved.
BuildProduction	Returns Yes if the database server is compiled for production use or returns No if the database server is a debug build.
BuildReproducible	Reserved.
BytesReceived	Returns the number of bytes received during client/server communications.

Property	Description
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.
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.
CacheHitsEng	Returns the number of database page lookups.
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.
CacheReadEng	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 server option" on page 134.
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.
CollectStatistics	Returns Yes or No to indicate whether the database server is collecting performance statistics. See "-k server option" on page 155.

Property	Description
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.
ConnsDisabled	Returns Yes or No to indicate the current setting of the server option to disable new connections. ☞ For information, see "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].
ConsoleLogFile	Returns the name of the file where messages from the Server Messages window are logged if the -o option was specified, otherwise returns an empty string. See "-o server option" on page 161.
ConsoleLogMaxSize	Returns the maximum size in bytes of the file used to log the Server Message window information. See "-os server option" on page 162.
CurrentCacheSize	Returns the current cache size, in kilobytes.
DebuggingInformation	Returns Yes if the server is displaying diagnostic messages for troubleshooting, and No otherwise. See "-z server option" on page 187.
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).
DiskReadEng	Returns the number of disk reads.
ExchangeTasks	Returns the number of tasks used for parallel execution of queries.
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 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.

Property	Description
FunctionMinParms	Returns the minimum number of parameters that must be specified 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.
HttpPorts	Returns the TCP/IP port numbers for the web server.
HttpsPorts	Returns the TCP/IP port number for the web server.
IdleTimeout	Returns the default idle timeout. See "-ti server option" on page 175.
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	Returns Yes if the server is an IQ server, and No otherwise.
IsNetworkServer	Returns Yes if connected to a network database server, and No if connected to a personal database server.
IsRsaAvailable	Returns Yes if the RSA DLL is installed, and No otherwise.
IsRuntimeServer	Returns Yes if connected to the limited desktop runtime database server, 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.

Property	Description
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.
LivenessTimeout	Returns the client liveness timeout default. See "-tl server option" on page 175.
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 rate at which database page address space windows are being mapped to physical memory in the cache using Address Windowing Extensions.
MaxCacheSize	Returns the maximum allowed cache size, 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 server option" on page 149. Computer resources typically limit the number of connections to a network server to a lower value than the default.
MaxMessage	Returns the current maximum line number that can be retrieved from the Server Messages window. This represents the most recent message displayed in the Server Messages window.

Property	Description
Message, *linenumber*	Returns a line from the 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 Server Messages window. Calling `PROPERTY( "message", i )` returns the i-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.
MessageText, *linenumber*	Returns the text associated with the specified line number in the server's message window, without a date and time prefix. The second parameter specifies the line number.
MessageTime, *linenumber*	Returns the date and time associated with the specified line number in the server's message window. The second parameter specifies the line number.
MessageWindowSize	Returns the maximum number of lines that can be retrieved from the server's message window.
MinCacheSize	Returns the minimum allowed cache size, in kilobytes.
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 server option" on page 149.
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 database option" on page 200.

Property	Description
NativeProcessorArchitecture	Returns a string that identifies the native processor type on platforms where a processor can be emulated (such as X86 on Win64). In all other cases, it returns the same value as property('ProcessorArchitecture'). Values can include: ♦ 32-bit Windows, except Windows CE, - X86 ♦ Windows CE - ARM ♦ 64-bit Windows - IA64 or X86_64 ♦ NetWare - X86 ♦ Solaris - SPARC, X86, or X86_64 ♦ AIX - PPC ♦ MAC OS - PPC ♦ HP - PA_RISC or IA64 ♦ Linux - X86, IA64, or X86_64 For a full list of supported platforms, see http://www.ianywhere.com/products/supported_platforms.html.
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 XP/200x, use the -gtc option to change the number of logical processors used. See "-gtc server option" on page 153.
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.
NumPhysicalProcessorsUsed	Returns the number of physical processors the database server will use. The personal server is limited to one processor on some platforms. On Windows XP/200x, you can use the -gt option to change the number of physical processors used by the network database server. See "-gt server option" on page 152.
OmniIdentifier	This property is reserved for system use. Do not change the setting of this option.
PacketsReceived	Returns the number of client/server communication packets received.
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.
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.)

Property	Description
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 2000, this property returns `Windows2000`.
PlatformVer	Returns the operating system on which the software is running, including build numbers, service packs, and so on. For example, it could return `Windows 2000 Build 2195 Service Pack 3`.
ProcessCPU	Returns CPU usage statistics for the server process. Values are in seconds. This property is supported on Windows XP/200x and Unix. This property is not supported on Windows CE or NetWare. 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.
ProcessCPUSystem	Returns process CPU system usage. Values are in seconds. This property is supported on Windows XP/200x and Unix. This property isn't supported on Windows CE or NetWare. 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 process CPU user usage. Values are in seconds. This property is supported on Windows XP/200x and Unix. This property isn't supported on Windows CE or NetWare. 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. Values can include: ♦ 32-bit Windows (not CE) - X86 ♦ 64-bit Windows - IA64 or X86_64 ♦ CE - ARM ♦ NetWare - X86 ♦ Solaris - SPARC, X86, or X86_64 ♦ AIX - PPC ♦ MAC OS - PPC ♦ HP - PA_RISC or IA64 ♦ Linux - X86, IA64, or X86_64

Property	Description
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. ☞ For more information, see “sa_server_option system procedure” [*SQL Anywhere Server - SQL Reference*].
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. ☞ For more information, 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).
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 server option” on page 190.
RememberLastStatement	Returns Yes if the server is recording the last statement prepared by each connection, and No otherwise. See “-zl server option” on page 187.
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. ☞ For information, see “sa_server_option system procedure” [*SQL Anywhere Server - SQL Reference*].

Property	Description
RequestLogging	Returns one of SQL, PLAN, HOSTVARS, PROCEDURES, TRIGGERS, OTHER, BLOCKS, REPLACE, ALL, or NONE, indicating the current setting for request logging. ☞ For information, see "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].
RequestLogMaxSize	Returns the maximum size of the request log file. See "-zs server option" on page 192.
RequestLogNumFiles	Returns the number of request log files being kept. ☞ For more information, see "sa_server_option system procedure" [*SQL Anywhere Server - SQL Reference*].
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 server option" on page 192.
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.
SendFail	Returns the number of times that the underlying communications protocols have failed to send a packet.
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. ☞ For more information about database mirroring, see "Understanding database mirroring" on page 828.
StartDBPermission	Returns the setting of the -gd server option, which can be one of DBA, all, or none. ☞ For more information, see "-gd server option" on page 146.
StartTime	Returns the date/time that the server started.
TempDir	Returns the directory in which temporary files are stored by the server.
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 thread.

Database-level properties

The following table lists properties available for each database on the server.

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_EXTENDED_PROPERTY function [System]” [*SQL Anywhere Server - SQL Reference*]
- “Server-level properties” on page 496
- “Connection-level properties” on page 476

Descriptions

Property	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.
AlternateServerName	Returns the alternate server named associated with the database if one was specified. See “-sn database option” on page 200.
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. ☞ For more information about database mirroring, see “Understanding database mirroring” on page 828.
AuditingTypes	Returns the types of auditing currently enabled. See “auditing option [database]” on page 395

Property	Description
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.
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*].
CharSet	Returns the CHAR character set of the database.
CheckpointLogBitmap-PagesWritten	Returns the number of writes to the checkpoint log bitmap.
CheckpointLogBitmapSize	Returns the checkpoint log bitmap size.
CheckpointLogCommitToDisk	Returns the number of checkpoint log commits to disk.
CheckpointLogPageInUse	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 rate at which the pre-images of database pages 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.

Property	Description
Checksum	Returns On if database page checksums are enabled for the database. 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.
Collation	Returns the collation used by the database. ☞ For a list of available collations, see "Supported and alternate collations" on page 328.
CommitFile	Returns the number of times the server has forced a flush of the disk cache. On Windows XP/200x and NetWare platforms, 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.
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 XP/200x.

Property	Description
DiskRead	Returns the number of pages that have been read from disk.
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.
DiskWrite	Returns the number of modified pages that have been written to disk.
DriveType *dbspace*	Returns the type of drive on which the database file is located. The value is one of the following: CD, FIXED, RAMDISK, REMOTE, REMOVABLE, or UNKNOWN. On Unix, depending on the version of Unix and the type of drive, it may not be possible to determining the drive type. In these cases "UNKNOWN" is returned. When used with DB_EXTENDED_PROPERTY, you can specify which dbspace you want the size for. The value *dbspace* can be either the *name* of the dbspace or the *file_id* of the dbspace. Leaving *dbspace* unspecified or using *system* both refer to the system dbspace. If the specified dbspace doesn't exist, the property function returns NULL. If the name of a dbspace is specified and the ID of a database that isn't the database of the current connection is also specified, the function also returns NULL.
Encryption	Returns the type of encryption used for database or table encryption, one of None, Simple, AES, or AES_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.

Property	Description
ExprCacheLookups	Returns the number of lookups performed in the expression cache.
ExprCacheResumesOfRead-Write	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 *dbspace*	Returns the file name of the database root file, including path.
FileSize *dbspace*	Returns the file size of the system dbspace, in pages, when used with the DB_PROPERTY function. When used with the DB_EXTENDED_PROPERTY function, you can specify which dbspace you want the size for. *Dbspace* can be either the *name* of the dbspace, the *file_id* of the dbspace, or *temporary* to refer to the temporary dbspace. You can also specify *translog* to return the size of the log file. Leaving the *dbspace* unspecified, or using *system*, both refer to the system dbspace. If the specified dbspace doesn't exist, the property function returns NULL. If the name of a dbspace is specified and an ID or name of a database that isn't the database of the current connection is also specified, the function also returns NULL.
FreePages *dbspace*	The FreePages property is only supported on databases created with version 8.0.0 or later. When used with the DB_PROPERTY function, this property returns the number of free pages in the system dbspace. When used with DB_EXTENDED_PROPERTY, you can specify which dbspace you want the number of free pages for. The value *dbspace* can be either the *name* of the dbspace, the *file_id* of the dbspace, or *temporary* to refer to the temporary dbspace. You can also specify *translog* to return the number of free pages in the log file. Leaving the *dbspace* unspecified, or using *system* both refer to the system dbspace. If the specified dbspace doesn't exist, the property function returns NULL. If the name of a dbspace is specified and an ID or name of a database that isn't the database of the current connection is also specified, the function also returns NULL.

Property	Description
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.
HashForcedPartitions	Returns the number of times that a hash operator was forced to partition because of competition for memory.
HashRowsFiltered	Returns the rate at which probe rows are rejected by bit-vector filters.
HashRowsPartitioned	Returns the rate at which rows are written to hash work tables.
HashWorkTables	Returns the number of work tables created for hash-based operations.
IdentitySignature	Reserved.
IdleCheck	Returns the number of times that the server's idle thread has become active to do idle writes, idle checkpoints, and so on.
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 100ths 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.
IOToRecover	Returns the estimated number of I/O operations required to recover the database.
IQStore	IQ Store is on/off. The value returned is always Off for SQL Anywhere.
JavaVM	Returns the Java VM the database server uses to execute Java in the database.

Property	Description
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 314.
LockCount	Returns the number of locks held by the connection.
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 XP/200x.
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.
LogWrite	Returns the number of pages that have been written to the transaction log.
LTMGeneration	Returns the generation number of the LTM or Replication Agent. This property is primarily for use by technical support.
LTMTrunc	Returns the minimal confirmed log offset for the Replication Agent.
MapPages	Returns the number of map pages used for accessing the lock table, frequency table, and table layout.
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.
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 of the primary'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. ☞ For more information about database mirroring, see "Understanding database mirroring" on page 828.

Property	Description
MultiByteCharSet	Returns On if the database uses a multibyte character set. Otherwise, returns Off.
Name	Returns the database name (identical to Alias).
NcharCharSet	Returns the NCHAR character set of the database.
NcharCollation	Returns the name of the collation used for NCHAR data.
NextScheduleTime *event-name*	Used with DB_EXTENDED_PROPERTY to return the next scheduled execution time for a specified event.
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.
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. ☞ For more information about database mirroring, see "Understanding database mirroring" on page 828.
PreserveSource	Returns On. This property is deprecated.
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.
QueryCachedPlans	Returns the number of cached execution plans across all connections.
QueryCachePages	Returns the number of pages used to cache execution plans.
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 available than the optimizer estimated, or because the execution plan required more memory than the optimizer estimated.

Property	Description
QueryOptimized	Returns the number of requests fully optimized.
QueryRowsBufferFetch	Returns the number of rows fetched using buffering.
QueryRowsMaterialized	Returns the rate at which rows are 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 server option” on page 151.
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.
RollbackLogPages	Returns the number of pages in the rollback log.
SendingTracingTo	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 connection.

Property	Description
SnapshotIsolationState	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. ☞ For more information, see "allow_snapshot_isolation option [database]" on page 389.
SortMergePasses	Returns the number of merge passes used during sorting.
SortRowsMaterialized	Returns the rate at which rows are 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.
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.
UniqueIdentifier	Returns On. This property is deprecated.
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.
XPathCompiles	Returns the number of times any XPath query (using the openxml procedure) was compiled by the database server since database server startup.

CHAPTER 14

Physical Limitations

Contents

About this chapter

This chapter describes the limitations on size and number of objects in SQL Anywhere databases.

Size and number limitations

The following table lists the physical limitations on size and number of objects in a SQL Anywhere database. The memory, CPU, and disk drive of the computer are more limiting factors in most cases.

Item	Limitation
Database size	13 files per database. For each file, the largest file allowed by operating system and file system
Dbspace size	2^{28} × page size
Temporary file size	2^{28} × 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, NetWare (NSS volumes), 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
NetWare (traditional volumes)	4 GB
File size (all other platforms and file systems)	2 GB
Maximum cache size (non-AWE cache) (Windows 2000 Professional, Windows 2000 Server, Windows XP Home Edition, Windows XP Professional, Windows Server 2003 Web Edition, Windows Server 2003 Standard Edition)	1.8 GB
Maximum cache size (non-AWE cache) (Windows 2000 Advanced Server, Windows 2000 Enterprise Server, Windows 2000 Datacenter Server, Windows Server 2003 Enterprise Edition, Windows Server 2003 Datacenter Edition)	2.7 GB
Maximum cache size (AWE cache) (Windows 2000 Professional, Windows 2000 Server, Windows 2000 Advanced Server, Windows 2000 Datacenter Server, 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 CE)	Limited by available memory on the device

Item	Limitation
Maximum cache size (Unix—Solaris, x86 Linux, AIX, HP)	2 GB for 32-bit servers
Maximum cache size (Win 64)	Limited by physical memory on 64-bit servers
Maximum cache size (Unix—Itanium Linux, Itanium HP-UX)	Limited by physical memory on 64-bit servers
Maximum cache size (NetWare)	2 GB
Maximum index entry size	No limit
Number of databases per server	255
Number of columns per table	♦ ((page size)/4)^2 for 32-bit servers ♦ ((page size)/8)^2 for 64-bit servers Note: An excessive number of columns, although allowed, will affect performance.
Number of indexes per table	2^{32}
Number of rows per database	4096 × 2^{28} ×13
Number of rows per table	4096 × 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 ×*number-of-database-connections*+65534
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

Item	Limitation
Database server names	♦ **TCP/IP** 250 bytes ♦ **Shared memory** 250 bytes ♦ **SPX** 32 bytes ☞ For more information, see “-n server option” on page 159 and “EngineName connection parameter [ENG]” on page 225.
DEFAULT or COMPUTE expressions (for example, in an ALTER TABLE statement)	Limited by page size (page size - 64 bytes)

Part III. Administering Your Database

This section describes how to use the tools included with SQL Anywhere to administer your database.

CHAPTER 15

SQL Anywhere Administration Tools

Contents

About this chapter

This chapter describes the graphical administration tools included with SQL Anywhere for administering your SQL Anywhere database. Some of these tools can also be used to administer UltraLite databases, MobiLink synchronization, and QAnywhere messaging.

Sybase Central

Sybase Central is a graphical tool for managing databases and related products. It helps you manage your 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 access these commands from the popup menu that appears when you right-click an object.
- **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 simply click and drag them to that location.
- **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 530.
- **Plug-in support** You can manage a variety of database products and tools by using plug-ins. Each plug-in provides its own online help, separate from this online help.

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 10 includes Sybase Central plug-ins for the following products:

- SQL Anywhere databases
- UltraLite databases
- MobiLink synchronization
- QAnywhere messaging

The plug-in files are found in the following location in your SQL Anywhere 10 installation:

Plug-in	File name and location
SQL Anywhere 10	*install-dir\java\saplugin.jar*
MobiLink 10	*install-dir\java\mlplugin.jar*
UltraLite 10	*install-dir\ultralite\java\ulplugin.jar*
QAnywhere 10	*install-dir\java\qaplugin.jar*

☞ For information about using the plug-ins included with SQL Anywhere 10, see:

- SQL Anywhere: "Using the SQL Anywhere plug-in" on page 535
- MobiLink: "MobiLink Plug-in for Sybase Central" [*MobiLink - Getting Started*]
- QAnywhere: "QAnywhere plug-in" [*QAnywhere*]
- UltraLite: "Creating an UltraLite database from Sybase Central" [*UltraLite - Database Management and Reference*] and "Working with UltraLite Databases" [*UltraLite - Database Management and Reference*]

Subject to your license agreement, you can deploy a set of 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 connecting to the sample database on Windows and Unix.

♦ To start Sybase Central and connect to the sample database (Windows)

1. From the Start menu, choose Programs ► SQL Anywhere 10 ► Sybase Central.

   Sybase Central opens.

2. In the Welcome to Sybase Central dialog, click View and Edit the Schema or Perform Maintenance on a Database.

   If the Welcome to Sybase Central dialog does not appear, choose Connections ► Connect with SQL Anywhere 10.

   The Connect dialog appears.

3. On the Identification tab, select ODBC Data Source Name and then type **SQL Anywhere 10 Demo**.

4. Click OK to 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 10.

♦ To start Sybase Central and connect to the sample database (Linux Desktop icons)

1. Open the SQL Anywhere 10 folder on your desktop.

2. Double-click Sybase Central.

   Sybase Central opens.

3. In the Welcome to Sybase Central dialog, click View and Edit the Schema or Perform Maintenance on a Database.

If the Welcome to Sybase Central dialog does not appear, choose Connections ► Connect with SQL Anywhere 10.

The Connect dialog appears.

4. On the Identification tab, select ODBC Data Source Name and then type **SQL Anywhere 10 Demo**.

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 278.

♦ **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. In the Welcome to Sybase Central dialog, click View and Edit the Schema or Perform Maintenance on a Database.

   If the Welcome to Sybase Central dialog does not appear, choose Connections ► Connect with SQL Anywhere 10.

   The Connect dialog appears.

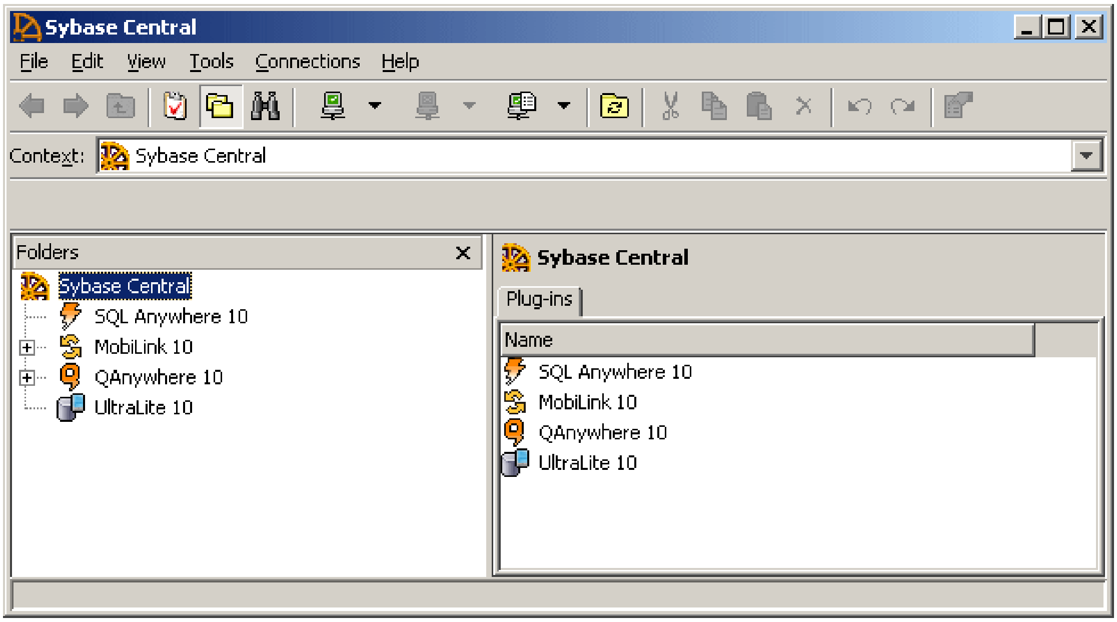
3. On the Identification tab, select ODBC Data Source Name and then type **SQL Anywhere 10 Demo**.

Navigating Sybase Central

This section explains how to navigate the Sybase Central user interface.

The Sybase Central main window

The Sybase Central main window:



Panes

The Sybase Central main window is split into two vertically-aligned panes.

You can choose whether you want the left pane to display a hierarchical view of database objects or the Task list for the currently-selected database object.

♦ To view the Task list or tree view in the left pane

1. Start Sybase Central.
2. From the View menu, choose Tasks or Folders to view the Task list or tree view, respectively.

The right pane has tabs that display the contents of the container that is selected in the left pane, as well as other information about the selected container. You can change the appearance of both panes in the Options dialog (accessed through the Tools menu).

You can configure the columns that appear on a tab in the right pane by choosing View ► Choose Columns.

Toolbar

The main window toolbar provides you with buttons for common commands. To show or hide the toolbar, choose Toolbar from the View menu. With the main toolbar, you can:

- navigate through the object tree
- connect to or disconnect from a database, server, or product plug-in
- access the Connection Profiles dialog (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 property sheet for a selected object

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.

Context dropdown list

The Context dropdown list, which appears below the toolbar, lets you navigate the object tree for a plug-in.

Using connection profiles

When you first connect to a server or database, you typically enter a user name, password, and other connection parameters.

To make subsequent connections from Sybase Central easier, you can create a connection profile that saves a group of parameters under a name that you supply. From then on, you can use that profile to connect to the object in one step.

To use and manage connection profiles, choose Connections ► Connection Profiles. This command opens the Connection Profiles dialog, where you can:

- create a new connection profile
- connect using a connection profile
- edit an existing connection profile
- set a profile to connect automatically when Sybase Central is started
- import or export a connection profile
- delete (or remove) profiles

Note
Connection profiles are specific to Sybase Central. If you are building an ODBC application, you can achieve functionality similar to connection profiles using ODBC data sources. See "Saving connection parameters in ODBC data sources" on page 53.

♦ To create a new connection profile

1. In Sybase Central, choose Connections ► Connection Profiles.

   The Connection Profiles dialog appears.

2. Click New.
3. In the New Connection Profile dialog, enter a name for the new profile and choose the appropriate plug-in. The plug-in is the product, such as SQL Anywhere 10 or MobiLink 10.
4. If desired, allow other users to access the profile by selecting Share This Connection Profile With Other Users. This is useful on multi-user platforms such as Unix.
5. Click OK.

   The Edit Connection Profile dialog appears.
6. In the Edit Connection Profile dialog, enter the desired values, and then click OK to close the dialog.

♦ To connect automatically when Sybase Central starts

1. In Sybase Central, choose Connections ► Connection Profiles.

   The Connection Profiles dialog appears.
2. Select a connection profile.
3. Click Set Startup to change the setting in the Use On Startup column to Yes.

♦ To edit the parameters of an existing connection profile

1. In Sybase Central, choose Connections ► Connection Profiles.

   The Connection Profiles dialog appears.
2. Select a connection profile.
3. Click Edit.

   The Edit Connection Profile dialog appears.
4. In the Edit Connection Profile dialog, edit the desired values.

♦ To import a connection profile

1. In Sybase Central, choose Connections ► Connection Profiles.

   The Connection Profiles dialog appears.
2. Click Import.

   The Import Connection Profile File dialog appears.
3. Specify the name of the file containing the connection profile you want to import.
4. Click OK.

♦ To export a connection profile

1. In Sybase Central, choose Connections ► Connection Profiles.

   The Connection Profiles dialog appears.

2. Select the connection profile you want to export.

3. Click Export.

   The Save As dialog appears.

4. Enter a name for the file to which you want to save the connection profile.

5. Click Save.

Sybase Central keyboard shortcuts

Sybase Central provides the following keyboard shortcuts.

Function key	Description
Alt+Enter	Opens the Properties dialog for the selected item.
Ctrl+C	Copies the selection to the clipboard.
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 dialog.
F11	Opens the Connection menu if there are multiple plug-ins loaded. If only one plug-in is loaded, this opens the connection dialog 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 dialog where you can select the connection you wish to disconnect.
Shift+F10	Opens popup menu.

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 where you can display, edit, and print code and messages.

☞ For information about using the SQL Statements pane, see "Interactive SQL main window description" on page 541, and "Interactive SQL keyboard shortcuts" on page 556.

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

☞ For a list of keyboard shortcuts that can be used in the Code Editor, see "Code Editor keyboard shortcuts" on page 531.

Customizing the Code Editor

You can customize the display characteristics of the Code Editor using the Options dialog. This dialog lets you change settings for the foreground and background colors, as well as the overall Code Editor appearance. All changes you make persist between sessions.

♦ To set Code Editor settings when editing on the SQL tab

1. In the Code Editor, choose Tools ► Options.
2. Configure the settings on the various tabs. Click OK.

♦ To set Code Editor settings when editing in a separate window

1. From the File menu, choose Customize Editor.
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, this 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+Del	Deletes the word to the right of the cursor.
Ctrl+End	Moves the cursor to the bottom of the Code Editor window.
Ctrl+F	Finds the specified text.
Ctrl+F3	Finds the next occurrence of the currently-selected text.
Ctrl+G	Opens the Go To dialog where you can specify the location you want to go to.
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 does not apply when editing text on the SQL tab in the right pane of Sybase Central.
Ctrl+O	Opens a file. This shortcut does not apply when editing text on the SQL tab in the right pane of Sybase Central.
Ctrl+P	Prints the contents of the SQL Statements pane. You can configure the appearance of the printed text in the Interactive SQL Options dialog.
Ctrl+Right Arrow	Moves the cursor forward one word.
Ctrl+S	Saves the contents of the SQL Statements pane.
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.

Function key	Description
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.
Ctrl+Shift+U	Changes the selection to uppercase characters.
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.
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 dialog 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 dialog 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 context 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 dialog 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

1. In Sybase Central, 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.

Using the SQL Anywhere plug-in

You can use the SQL Anywhere plug-in to upgrade existing databases, create new databases, and administer databases. You can choose the desired mode from the Mode menu or by clicking the toolbar button for the mode.

The SQL Anywhere plug-in can operate in any of the following modes:

- **Design mode** While operating in Design mode, you can create and modify database objects such as tables, users, triggers, indexes, remote 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 operating in Debug mode, you can use the SQL Anywhere debugger to assist you in developing SQL stored procedures, triggers, and event handlers.

  ☞ For information about using Debug mode, see “Debugging Procedures, Functions, Triggers, and Events” [*SQL Anywhere Server - SQL Usage*].

- **Application Profiling mode** While operating 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 information about using Application Profiling mode, see “Application profiling” [*SQL Anywhere Server - SQL Usage*].

See also

- “Connecting from Sybase Central, Interactive SQL, or the SQL Anywhere Console utility” on page 54
- “Working with Database Objects” [*SQL Anywhere Server - SQL Usage*]
- “Creating a Windows CE database using Sybase Central” on page 977

Copying database objects in the SQL Anywhere plug-in

In the SQL Anywhere 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.

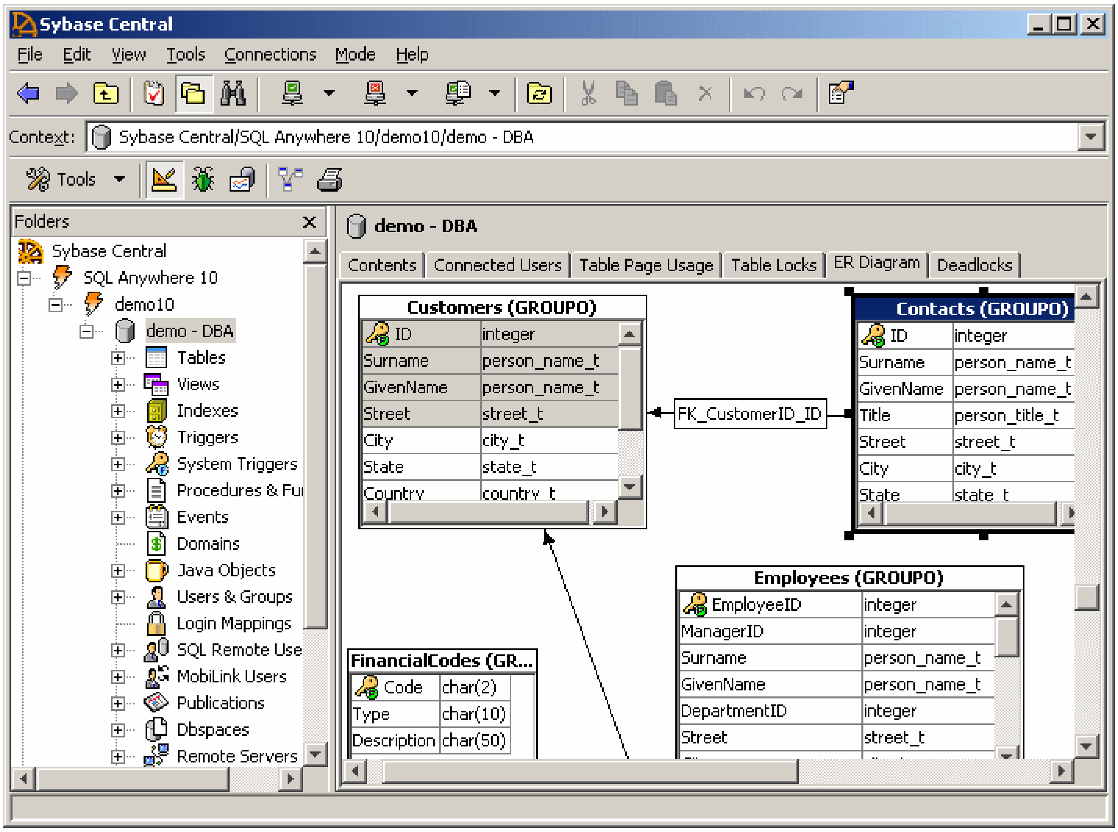
You can also paste objects onto other objects in the database. For example, if you paste a table onto a user, this gives the user permissions on the table.

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. 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 plug-in:

- Articles
- Check constraints
- Columns
- Dbspaces
- Directory access servers
- Domains
- Events
- External logins
- Foreign keys
- Indexes
- Login mappings (integrated logins and Kerberos logins)
- Maintenance plans
- Maintenance plan reports
- Message types
- MobiLink users
- Primary keys
- Procedures and functions
- Publications
- Remote servers
- Schedules
- SQL Remote subscriptions
- Synchronization subscriptions
- System triggers
- Tables
- Triggers
- Unique constraints
- Users and groups
- Views
- Web services

Viewing entity-relationship diagrams from the SQL Anywhere plug-in

When you are connected to a database from the SQL Anywhere plug-in, you can view an entity-relationship diagram of the tables in the database. When you have the database selected, click the ER Diagram tab in the right pane to see the diagram.



Customers (GROUPO)

ID	integer
Surname	person_name_t
GivenName	person_name_t
Street	street_t
City	city_t
State	state_t
Country	country_t

Contacts (GROUPO)

ID	integer
Surname	person_name_t
GivenName	person_name_t
Title	person_title_t
Street	street_t
City	city_t
State	state_t

FinancialCodes (GR...

Code	char(2)
Type	char(10)
Description	char(50)

Employees (GROUPO)

EmployeeID	integer
ManagerID	integer
Surname	person_name_t
GivenName	person_name_t
DepartmentID	integer
Street	street_t

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 or from the toolbar dropdown list, and then choose File ▶ Filter Objects by Owner.

   Choose the database users whose tables you want to see in the entity-relationship diagram.

2. Click the ER Diagram tab in the right pane.

   The entity-relationship diagram appears.

3. Choose File ▶ Choose ER Diagram Tables.

   The Choose ER Diagram Tables dialog appears. This dialog lets you further customize the tables that appear in the entity-relationship diagram.

   Use the Add and Remove buttons to customize the tables that appear in the Selected Tables list.

See also

- “Entity-relationship diagrams” [*SQL Anywhere Server - SQL Usage*]

Interactive SQL

Interactive SQL is a tool shipped 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 the following purposes:

- Sending SQL statements to the database server. See "Executing SQL statements from Interactive SQL" on page 543.
- Browsing the information in a database. See "Executing SQL statements from Interactive SQL" on page 543.
- Editing data in result sets. See "Editing result sets in Interactive SQL" on page 548.
- Loading data into a database. See "Using the Interactive SQL Import wizard" [*SQL Anywhere Server - SQL Usage*].
- Exporting query results to a file. See "Exporting query results" [*SQL Anywhere Server - SQL Usage*].
- Running command files or script files. See "Running SQL command files" [*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 "Query Editor Help" [*SQL Anywhere 10 - Context-Sensitive Help*].

Interactive SQL is available on Windows XP/200x, Solaris, Linux, and Mac OS X.

SQL statements used only from Interactive SQL

Interactive SQL supports all SQL statements supported by SQL Anywhere and UltraLite databases, as well as several SQL statements that can be used only from 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 ENGINE 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*]

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 prompt)

- Execute a `dbisql` command from a command prompt.

  If you omit the -c option, which specifies the connection parameters for the database, or if you supply insufficient connection parameters, the Connect dialog appears, where you can enter connection information for the database. For information about the supported options, see “Interactive SQL utility (dbisql)” on page 622.

  The following command starts Interactive SQL and connects to the sample database:

  ```
  dbisql -c "UID=DBA;PWD=sql;DSN=SQL Anywhere 10 Demo"
  ```

♦ To start Interactive SQL (Windows)

1. From the Start menu, choose Programs ► SQL Anywhere 10 ► Interactive SQL.

   Interactive SQL opens, and the Connect dialog appears.

2. Enter the connection information for your database in the Connect dialog.

♦ To start Interactive SQL (Sybase Central)

1. From the Tools menu, choose SQL Anywhere 10 ► Open Interactive SQL.

   Interactive SQL opens, and the Connect dialog appears.

2. Enter the connection information for your database in the Connect dialog.

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 10.

♦ To start Interactive SQL (Linux Desktop icons)

1. Open the SQL Anywhere 10 folder on your desktop.

2. Double-click Interactive SQL.

   Interactive SQL opens, and the Connect dialog appears.

3. Enter the connection information for your database in the Connect dialog.

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 278.

♦ To start Interactive SQL (Unix command line)

1. In a terminal session, enter the following command:

   ```
   dbisql
   ```

   Interactive SQL opens, and the Connect dialog appears.

2. Enter the connection information for your database in the Connect dialog.

Tip
You can also access Interactive SQL from within Sybase Central in the following ways:

- Selecting a database, and choosing Open Interactive SQL from the File menu.
- Right-clicking a database, and choosing Open Interactive SQL from the popup menu.
- Right-clicking a stored procedure, and choosing Execute from Interactive SQL from the popup menu. 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.

Interactive SQL main window description

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.
- **Results** The Results pane has three tabs: Results, Messages, and Plan. 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 548.

  The Messages tab displays messages from the database server about the SQL statements that you execute in Interactive SQL.

The Plan tab displays the query optimizer's execution plan for a SQL statement. See "Graphical plans" [*SQL Anywhere Server - SQL Usage*].

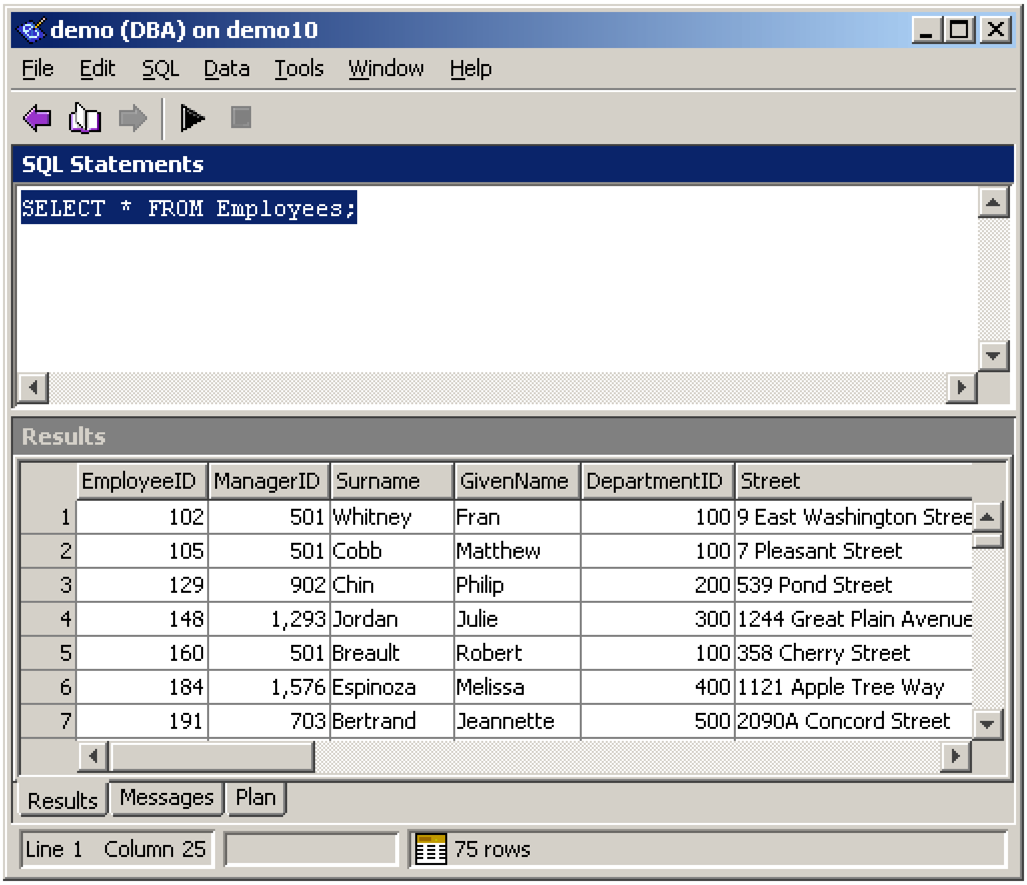
You can configure settings for the tabs and panes in Interactive SQL from the Options dialog, accessing by choosing Tools ► Options.

When you are connected to a database from Interactive SQL, the title bar at the top of the window displays connection information, as follows:

```
database-name ( userid ) on server-name
```

For example, if you connect to the sample database using the SQL Anywhere 10 Sample ODBC data source, the title bar contains the following information:

```
demo ( DBA ) on demo10
```



```
SELECT * FROM Employees;
```

	EmployeeID	ManagerID	Surname	GivenName	DepartmentID	Street
1	102	501	Whitney	Fran	100	9 East Washington Stree
2	105	501	Cobb	Matthew	100	7 Pleasant Street
3	129	902	Chin	Philip	200	539 Pond Street
4	148	1,293	Jordan	Julie	300	1244 Great Plain Avenue
5	160	501	Breault	Robert	100	358 Cherry Street
6	184	1,576	Espinoza	Melissa	400	1121 Apple Tree Way
7	191	703	Bertrand	Jeannette	500	2090A Concord Street

Line 1 Column 25

75 rows

Executing SQL statements from Interactive SQL

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.

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.

♦ To execute a SQL statement

1. Type your query in the Statements pane.
2. Click Execute SQL Statement, or choose SQL ► Execute, or press F5 to execute the statement.

♦ To clear the SQL Statements pane

- From the Edit menu choose Clear SQL, or press Escape.

Executing multiple 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. See "command_delimiter option [Interactive SQL]" on page 562.

An alternative to using the semicolon is to enter the separator **go** on a line by itself, at the beginning of the line.

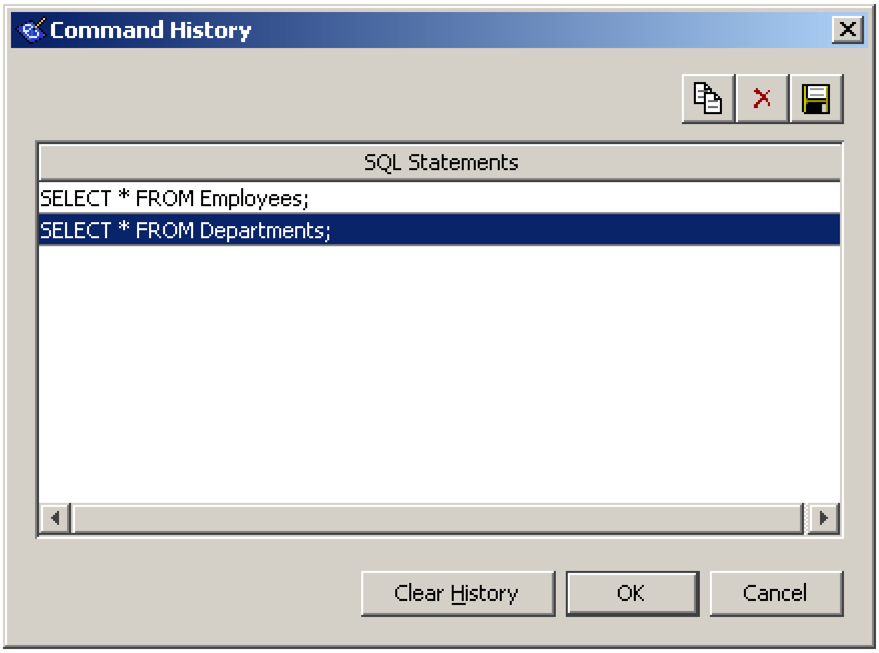
By default, Interactive SQL shows the first result set of the last statement executed. If you want to see all the result sets from the last statement, you can select the Show Multiple Result Sets option on the Results tab of the Options dialog.

Tip
You can press F9 to execute only the selected text in the SQL Statements pane.

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 dialog. To access the Command History dialog, press Ctrl+H, or click the book icon on the toolbar.



SELECT * FROM Employees;
SELECT * FROM Departments;

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 dialog.

You can also recall commands without the Command History dialog. Use the arrows in the toolbar to scroll back and forward through your commands, or press Alt+Right Arrow and Alt+Left Arrow.

Note
If you execute a SQL statement that contains password information (GRANT CONNECT, GRANT REMOTE DBA, CONNECT, or CREATE EXTERNLOGIN), the password information appears in the Command History dialog 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:

```
GRANT CONNECT TO testuser
 IDENTIFIED BY testpassword
```

the following appears in the Command History dialog in subsequent Interactive SQL sessions:

```
GRANT CONNECT TO testuser
 IDENTIFIED BY ...
```

Copying commands from the Command History dialog

You can copy commands from the Command History dialog to use elsewhere. When you copy multiple commands, they are separated by the command delimiter (a semicolon by default).

♦ To copy commands from the Command History dialog

1. Open the Command History dialog.
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 dialog

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 dialog.
2. Click the Save button or press Ctrl+S.
3. In the Save As dialog, specify a location and name for the file.

   The command history file has a *.sql* extension.
4. Click Save when finished.

Removing commands from the Command History dialog

The contents of the Command History dialog 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 dialog. This action cannot be undone.
- Remove all the commands from the dialog by clicking Clear History. This action cannot be undone.

Logging commands

With the Interactive SQL logging feature, you can record commands as you execute them. Interactive SQL continues to record 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.

♦ To begin logging Interactive SQL commands

1. From the SQL menu, choose Start Logging.
2. In the Save As dialog, specify a location and name for the log file. For example, name the file *filename.sql*.
3. Click Save when finished.

♦ To stop logging Interactive SQL commands

- From the SQL menu, choose Stop Logging.

Tips
You can also start and stop logging by typing in the SQL Statements pane. To start logging, type and execute **START LOGGING** *'c:\filename.sql'*, where *c:\filename.sql* is the path, name, and extension of the log file. You only need to include the single quotation marks if the path contains embedded spaces. To stop logging, type and execute **STOP LOGGING**.
Once you start logging, all commands that you try to execute are logged, including ones that do not execute properly.

Canceling commands in Interactive SQL

A Cancel operation stops the current processing and prompts for the next command. The Interrupt button on the Interactive SQL toolbar cancels a command.

If a command file was being processed, or if there is more than one statement in the SQL Statements pane, 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 571.

Using Interactive SQL with 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 export command files.

You can execute command files in any of the following ways from Interactive SQL:

- Using the Interactive SQL READ statement
- Choosing File ► Run Script
- Supplying a command file as a command line argument for Interactive SQL

On Windows platforms you can make Interactive SQL the default editor for *.sql* command files. This lets you double-click the file so that its contents appears in the SQL Statements pane of Interactive SQL.

♦ To make Interactive SQL the default editor for .sql files

1. From Interactive SQL, choose Tools ► Options.

   The Options dialog appears.

2. On the General tab, select the Make ISQL the Default Editor for .SQL Files and Plan Files option.

3. Click OK.

☞ For information about using Interactive SQL with command files, see:

- "Using SQL command files" [*SQL Anywhere Server - SQL Usage*]
- "Running SQL command files" [*SQL Anywhere Server - SQL Usage*]

- “Loading SQL command files” [*SQL Anywhere Server - SQL Usage*]
- “Writing database output to a file” [*SQL Anywhere Server - SQL Usage*]

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.

♦ 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 tables and procedures lookup dialogs, 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 iAnywhere 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 576.

Printing SQL statements

You can print the contents of the SQL Statements pane by pressing Ctrl+P or by choosing Print from the File menu. You can add a header or footer and configure other formatting options in the Interactive SQL Options dialog (on the Editor page, click the Print tab). See "Print tab" [*SQL Anywhere 10 - Context-Sensitive Help*].

☞ For information about printing graphical plans in Interactive SQL, see "Graphical plans" [*SQL Anywhere Server - SQL Usage*].

Editing result sets in Interactive SQL

Once you execute a query in Interactive SQL, you can 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. Interactive SQL supports editing, inserting, and deleting rows. Editing the result set has the same effect as executing UPDATE, INSERT, and DELETE statements.

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 of 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).

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 of the values within existing rows in database tables. You must have UPDATE permission on the columns being modified. 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 from the popup menu, or press F2 to edit the result set.

   A blinking cursor appears in the table cell containing the value.
4. 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.
5. 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.
6. Execute a COMMIT statement to make your changes to the table permanent.

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 from the popup menu.

   A new blank row appears with a blinking cursor in the first value in the row.
2. Enter the new value.

   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.
3. Press Tab to move to the next column.
4. Repeat steps 2 and 3 until all the column values are added.
5. 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
```

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.

   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 rows from an Interactive SQL result set

You can copy rows directly from the result set in Interactive SQL and then paste them into other applications. Copying rows copies both the column heading and table data into the clipboard. Copied data is comma-delimited and all of the strings are enclosed in single quotes.

♦ To copy rows from the Interactive SQL result set

1. Select the row(s) you want to copy. 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.

2. Press Ctrl+C to copy the selected row(s).

   The selected row(s), including their column headings, are copied to the clipboard. You can now paste them into other applications.

Copying individual values from the result set

You can copy a single value from the result set by selecting a value, right-clicking the result set and choosing Copy Cell from the popup menu. When you do this, no column headings are copied—only the data is copied, and no quoting is done.

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 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.

   The Connect dialog appears.

   > **Tip**
   > If the SQLCONNECT environment variable is set, or if you are already connected to a SQL Anywhere database, the server attempts to use this information to connect to a database before it prompts you for information. Likewise, if the ULCONNECT environment variable is set, or if you are already connected to an UltraLite database, the 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 dialog appears.

2. In the Connect dialog, enter connection options, and click OK to 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.

Source control integration

Interactive SQL can integrate 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), such as Microsoft's Visual SourceSafe. On Windows and all other platforms, you can also configure Interactive SQL to use source control products that do not support the SCC API. In that case, you provide Interactive SQL with 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, 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.

☞ For descriptions of the supported actions, see:

- "Opening source control projects from Interactive SQL" on page 554
- "Checking files out from Interactive SQL" on page 554
- "Checking in files from Interactive SQL" on page 555
- "Additional source control actions" on page 555

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. From the Tools menu, choose Options.

   The Options dialog appears.

2. Click Source Control in the left pane of the Options dialog.

3. Select Enable Source Control Integration to turn on the source control extensions.

4. Click OK.

Configuring other source control systems

♦ To configure Interactive SQL source control systems with a command line interface

1. From the Tools menu, choose Options.

The Options dialog appears.

2. Click Source Control in the left pane of the Options dialog.

3. Select Enable Source Control Integration to turn on the source control extensions.

4. Select Custom Source Control System.

5. Click Configure.

   The Custom Source Control Options dialog appears where you can provide Interactive SQL with a set of commands to run for each of the listed source control actions.

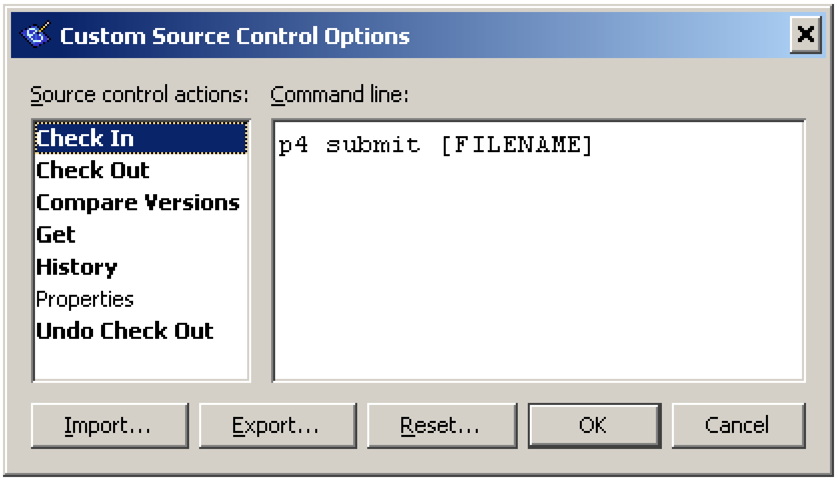
6. Click Reset.

   The Reset Source Control Commands dialog appears.

7. Select your source control system from the list, and then click OK.

8. 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.



```
p4 submit [FILENAME]
```

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 dialog (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 dialog. This may be useful if you need to configure Interactive SQL source control command lines on several computers.

9. 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 dialog 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

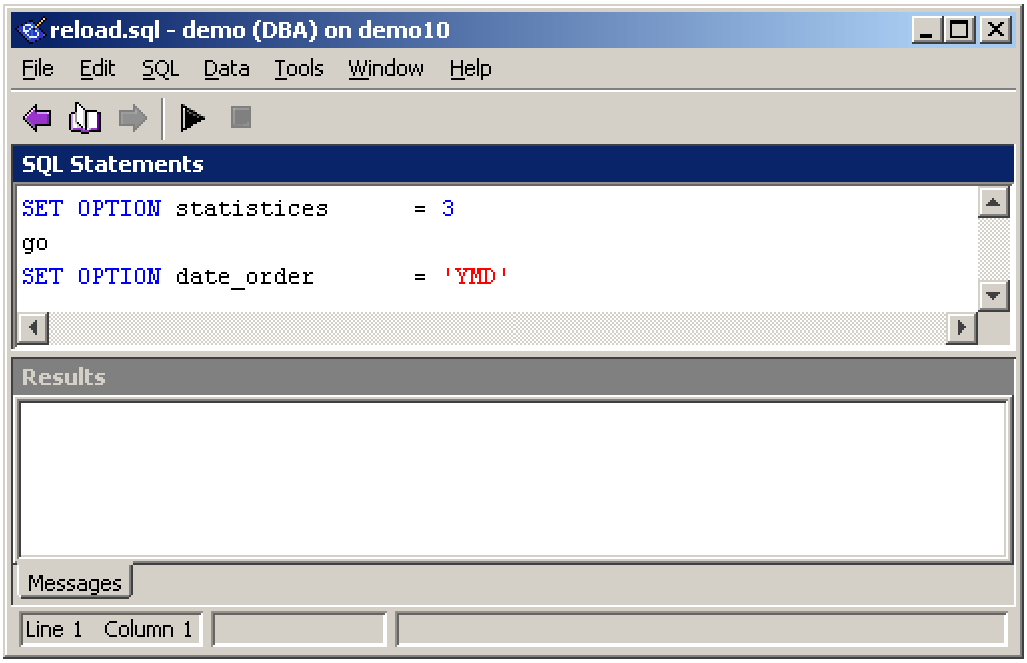
Once you open a file in Interactive SQL, there are two ways you can check the file out: modifying its contents in the SQL Statements pane, or using the command on the File menu.

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:


reload.sql - demo (DBA) on demo10
File Edit SQL Data Tools Window Help
SQL Statements
Results
Messages
Line 1 Column 1

```
SET OPTION statistices      = 3
go
SET OPTION date_order       = 'YMD'
```

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

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 throw away 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 Visual SourceSafe Explorer.

Interactive SQL keyboard shortcuts

Interactive SQL provides the following keyboard shortcuts:

Key(s)	Description
Alt+F4	Exits Interactive SQL.
Alt+Left cursor	Displays the previous SQL statement in the history list.
Alt+Right cursor	Displays the next SQL statement in the history list.
Ctrl+Break	Interrupts the SQL statement that is being executed.
Ctrl+A	Selects all text in the active pane.
Ctrl+C	Copies the selected row(s) and column headings to the clipboard in the Results pane. In the SQL Statements pane, copies the selected text to the clipboard.
Ctrl+End	Moves to the bottom of the current pane.
Ctrl+F	Opens the Find/Replace dialog.
Ctrl+G	Goes to the specified line in the SQL Statements pane.
Ctrl+H	Displays the history of your executed SQL.

Key(s)	Description
Ctrl+Home	Moves to the top of the current pane.
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. You can configure the appearance of the printed text in the Interactive SQL Options dialog. ☞ For information about setting print options, see "Print tab" [*SQL Anywhere 10 - Context-Sensitive Help*].
Ctrl+Q	Displays 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.
Ctrl+S	Saves the contents of the SQL Statements pane.
Ctrl+V	Pastes the selected text.
Ctrl+X	Cuts the selected text.
Ctrl+Y	Repeats the last operation.
Ctrl+Z	Undoes the last operation.
Esc	Clears the SQL Statements pane.
F1	Opens 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.
F5	Executes all text in the SQL Statements pane. You can also perform this operation by clicking Execute SQL Statement on the toolbar.
F7	Displays the Lookup Table Name dialog. In this dialog, 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 dialog. In this dialog, 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.

Key(s)	Description
F9	Executes the text that is selected in the SQL Statements pane. If no text is selected, all of the statements are executed.
F11	Opens the Connect dialog if Interactive SQL is not connected to a database.
F12	Disconnects from the database.
Page Down	Moves a page down in the current pane.
Page Up	Moves a page up in the current pane.
Shift+F5	Displays the plan for the statement in the SQL Statements pane without executing the statement.
Shift+F10	Displays the context menu for the area that has focus. This keyboard shortcut is an alternative to right-clicking an area.

The following keyboard shortcuts are available when the SQL Statements pane has the focus:

Key(s)	Description
Ctrl+]	Moves the cursor to the matching brace. Use this shortcut to match parentheses, braces, brackets, and angle brackets.
Ctrl+Backspace	Deletes the word to the left of the cursor.
Ctrl+Del	Deletes the word to the right of the cursor.
Ctrl+G	Opens the Go To dialog where you can specify the line you want to go to.
Ctrl+L	Deletes the current line from the SQL Statements pane and puts the line onto the clipboard.
Ctrl+Shift+]	Extends the selection to the matching brace. Use this shortcut to match parentheses, braces, brackets, and angle brackets.
Ctrl+Shift+L	Deletes the current line.
Ctrl+Shift+U	Changes the selection to uppercase characters.
Ctrl+U	Changes the selection to lower case characters.
F3	Finds the next occurrence of the selected text.
Home	Moves the cursor to the start of the current line or to the first word on the current line.
Shift+F3	Finds the previous occurrence of the selected text.
Shift+Home	Extends the selection to the start of the text on the current line.

Interactive SQL options

Syntax 1

SET [**TEMPORARY**] **OPTION** *option-name* = [*option-value*]

Syntax 2

SET PERMANENT

Syntax 3

SET

Parameters

userid: *identifier* or *string*

option-name: *identifier* or *string*

option-value: *string*, *identifier*, or *number*

Description

Syntax 1 stores the specified Interactive SQL option.

Syntax 2 stores all current Interactive SQL options.

Syntax 3 displays all of the current option settings.

Interactive SQL option settings are stored on the client computer. They are not stored in the database.

☞ For more information, see "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*].

Note
The syntax SET [TEMPORARY] OPTION [*userid* | PUBLIC.]*option-name* is deprecated. If you specify this syntax, the *userid* or PUBLIC keyword is ignored.

Option	Values	Default
"auto_commit option [Interactive SQL]" on page 561	On, Off	Off
"auto_refetch option [Interactive SQL]" on page 561	On, Off	On
"bell option [Interactive SQL]" on page 562	On, Off	On
"command_delimiter option [Interactive SQL]" on page 562	String	' ; '

Option	Values	Default
"commit_on_exit option [Interactive SQL]" on page 563	On, Off	On
"default_isql_encoding option [Interactive SQL]" on page 563	String	Empty string
"echo option [Interactive SQL]" on page 564	On, Off	On
"input_format option [Interactive SQL]" on page 565	ASCII, DBASE, DBASEII, DBASEIII, EXCEL, FIXED, FOXPRO, LOTUS	ASCII
"isql_command_timing option [Interactive SQL]" on page 566	On, Off	On
"isql_escape_character option [Interactive SQL]" on page 566	Character	' \ '
"isql_field_separator option [Interactive SQL]" on page 567	String	' , '
"isql_maximum_displayed_rows option [Interactive SQL]" on page 568	All or a non-negative integer	500
"isql_plan option [Interactive SQL]" on page 568	Short, Long, GraphicalLowDetail, GraphicalHighDetail	Graphical
"isql_print_result_set option [Interactive SQL]" on page 569	Last, All, None	Last
"isql_quote option [Interactive SQL]" on page 570	String	'
"isql_show_multiple_result_sets [Interactive SQL]" on page 570	On, Off	Off
"nulls option [Interactive SQL]" on page 571	String	'(NULL)'
"on_error option [Interactive SQL]" on page 571	Stop, Continue, Prompt, Exit, Notify_Continue, Notify_Stop, Notify_Exit	Prompt
"output_format option [Interactive SQL]" on page 572	ASCII, DBASEII, DBASEIII, EXCEL, FIXED, FOXPRO, HTML, LOTUS, SQL, XML	ASCII
"output_length option [Interactive SQL]" on page 573	Integer	0
"output_nulls option [Interactive SQL]" on page 574	String	Empty string

Option	Values	Default
"truncation_length option [Interactive SQL]" on page 574	Integer	256

auto_commit option [Interactive SQL]

Function

Controls whether a COMMIT is performed after each statement.

Allowed values

On, Off

Default

Off

Description

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).

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

auto_refetch option [Interactive SQL]

Function

Controls whether query results are fetched again after deletes, updates, and inserts.

Allowed values

On, Off

Default

On

Description

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]

Function

Controls whether the bell sounds when an error occurs.

Allowed values

On, Off

Default

On

Description

Set this option according to your preference.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

command_delimiter option [Interactive SQL]

Function

Sets the string that indicates the end of a statement in Interactive SQL.

Allowed values

String

Default

Semi-colon (;)

Description

In general, there is no need to change the command delimiter. You should leave it as a semicolon.

An alternative to using a semi-colon 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 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 parenthesis),) (right parenthesis), 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.

- 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.

See also

- "The Interactive SQL utility" on page 622
- "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]

Function

Controls behavior when Interactive SQL disconnects or terminates.

Allowed values

On, Off

Default

On

Description

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*]

default_isql_encoding option [Interactive SQL]

Function

Specifies the code page that should be used by READ, INPUT, and OUTPUT statements.

Allowed values

Identifier or string

Default

Use system code page (empty string)

Scope

Can be set as a temporary option only, for the duration of the current connection.

Description

This option is used to specify the code page to use when reading or writing files. It cannot be set permanently. The default code page is the default code page for the platform you are running on. On English Windows computers, the default code page is 1252.

Interactive SQL determines the code page that is used for a particular INPUT, OUTPUT, or READ statement as follows, where code page values occurring earlier in the list take precedence over those occurring later in the list:

- the code page specified in the ENCODING clause of the INPUT, OUTPUT, or READ statement
- the code page specified with the default_isql_encoding option (if this option is set)
- the code page specified with the -codepage option when Interactive SQL was started
- the default code page for the computer Interactive SQL is running on

☞ For more information about code pages and character sets, see "International language and character set tasks" on page 323.

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*]
- "Overview of character sets, encodings, and collations" on page 310
- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

Example

Set the encoding to UTF-16 (for reading Unicode files):

```
SET OPTION default_isql_encoding = 'UTF-16';
```

echo option [Interactive SQL]

Function

Controls whether statements are echoed to the log file before they are executed.

Allowed values

On, Off

Default

On

Description

This option is most useful when you use the READ statement to execute a Interactive SQL command file. Logging must be turned on in order 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]

Function

Sets the default data format expected by the INPUT statement.

Allowed values

String (see below for allowed values)

Default

ASCII

Description

Allowable input formats are:

- **ASCII** Input lines are assumed to be ASCII characters, one row per line, with values separated by commas. Alphabetic strings can be enclosed in apostrophes (single quotes) or quotation marks (double quotes). 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 than the default, which is a comma (,).

  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.

- **DBASE** The file is in dBASE II or dBASE III format. Interactive SQL attempts to determine which format, based on information in the file.

- **DBASEII** The file is in dBASE II format.

- **DBASEIII** The file is in dBASE III format.

- **EXCEL** Input file is in the format of Microsoft Excel 2.1.

- **FIXED** Input lines are in fixed format.

- **FOXPRO** The file is in FoxPro format.
- **LOTUS** The file is a Lotus WKS format worksheet. INPUT assumes that the first row in the Lotus WKS format worksheet consists of column names.

See also

- "INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

isql_command_timing option [Interactive SQL]

Function

Controls whether SQL statements are timed or not.

Allowed values

On, Off

Default

On

Description

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 tab after you execute a statement. If you set the option to Off, the time does not appear.

You can also set this option on the Messages tab of the Options dialog.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

isql_escape_character option [Interactive SQL]

Function

Controls the escape character used in place of unprintable characters in data exported to ASCII files.

Allowed values

Any single character

Default

A backslash (\ )

Description

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 an ASCII file.

See also

- ♦ "isql_quote option [Interactive SQL]" on page 570
- ♦ "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 ASCII;
```

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]

Function

Controls the default string used for separating values in data exported to ASCII files.

Allowed values

String

Default

A comma (,)

Description

Controls the default string used for separating (or delimiting) values in data exported to ASCII files. If an OUTPUT statement does not contain a DELIMITED BY clause, the value of this setting is used.

See also

- ♦ "isql_quote option [Interactive SQL]" on page 570
- ♦ "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

Example

Set 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 ASCII;
```

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.

isql_maximum_displayed_rows option [Interactive SQL]

Function

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

Description

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 dialog in Interactive SQL.

See also

♦ "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

isql_plan option [Interactive SQL]

Function

Controls the type of access plan that appears on the Plan tab in the Interactive SQL Results pane after you execute statements.

Allowed values

SHORT, LONG, GRAPHICAL, GRAPHICALLOWDETAIL, GRAPHICALHIGHDETAIL

Default

GRAPHICAL

Description

This option allows you to specify the type of access plan that appears for SELECT, INSERT, UPDATE, and DELETE statements. To set the level of detail for the plan the optimizer provides, you can choose one of the following values:

- **SHORT** Provides basic information about the access plan.
- **LONG** Provides detailed information about the access plan.
- **GRAPHICAL** Provides a graphical plan with no statistics.
- **GRAPHICALLOWDETAIL** Provides a tree diagram of the query with statistics for the root node of the query.
- **GRAPHICALHIGHDETAIL** Provides a tree diagram of the query, as well as statistics that indicate the resources used by the part of the query that is selected.

The plan is computed only when you click the Plan tab.

You can also specify the plan type on the Plan tab of the Interactive SQL Options dialog.

Note that the access plan that appears may not be the same plan that was used when the statement was executed. The plan is fetched after the SQL statement is executed, at which point the optimizer may have new information available. This means that the optimizer may return a different plan than the one it used to execute the statement.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

isql_print_result_set option [Interactive SQL]

Function

Specifies which result set(s) are printed when a *.sql* file is run.

The isql_print_result_set option takes effect only when you run the Interactive SQL (dbisql) utility within a command window (for example, when running a *.sql* file).

Allowed values

LAST, ALL, NONE

Default

LAST

Description

This option allows you to specify which result set(s) are printed when a *.sql* file is run.

You can choose one of the following print options:

- **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.

Although this option has no affect when Interactive SQL is running in windowed mode, you can still view and set the option in windowed mode. Choose Tools ▶ Options. On the Results tab, select the appropriate action in the Console Mode area.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

isql_quote option [Interactive SQL]

Function

Controls the default string that begins and ends all strings in data exported to ASCII files.

Allowed values

String

Default

A single apostrophe (')

Description

Controls the default string that begins and ends all strings in data exported to ASCII files. If an OUTPUT statement does not contain a QUOTE clause, this value is used by default.

See also

- "isql_field_separator option [Interactive SQL]" on page 567
- "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 ASCII;
```

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]

Function

Specifies whether multiple result sets can appear in the Results pane in Interactive SQL.

Allowed values

ON, OFF

Default

OFF

Description

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 in console (command-prompt) mode.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

nulls option [Interactive SQL]

Function

Specifies how NULL values in the database appear when displaying results in Interactive SQL.

Allowed values

String

Default

(NULL)

Description

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 574

on_error option [Interactive SQL]

Function

Controls what happens if an error is encountered while executing statements in Interactive SQL.

Allowed values

String (see below for allowed values)

Default

Prompt

Description

Controls what happens if an error is encountered while executing statements, as follows:

- **Stop** Interactive SQL stops executing statements.
- **Prompt** Interactive SQL prompts the user to see if the user wants to continue.
- **Continue** The error is ignored and Interactive SQL continues executing statements.
- **Exit** Interactive SQL terminates.
- **Notify_Continue** The error is reporting, 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 terminate Interactive SQL.

When you are executing a *.sql* file, the values Stop and Exit are equivalent. If you specify either of these values, Interactive SQL terminates.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

output_format option [Interactive SQL]

Function

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

String (see below for allowed values)

Default

ASCII

Description

The valid output formats are:

- **ASCII** The output is an ASCII 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) will be quoted.

  Three other special sequences are also used. The two characters **\n** represent a newline character; \\ represents a single backslash character, and the sequence **\xDD** represents the character with hexadecimal code DD.

- **DBASEII** The output is a dBASE II format file with the column definitions at the top of the file. Note that a maximum of 32 columns can be output. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.
- **DBASEIII** The output is a dBASE III format file with the column definitions at the top of the file. Note that a maximum of 128 columns can be output. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.
- **EXCEL** The output is an Excel 2.1 worksheet. The first row of the worksheet contains column labels (or names if there are no labels defined). Subsequent worksheet rows contain the actual table data.
- **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.
- **FOXPRO** The output is a FoxPro format file with the column definitions at the top of the file. Note that a maximum of 128 columns can be output. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.
- **HTML** The output is in HTML format.
- **LOTUS** The output is a Lotus WKS format worksheet. Column names will be put as the first row in the worksheet. Note that there are certain restrictions on the maximum size of Lotus WKS format worksheets that other software (such as Lotus 1-2-3) can load. There is no limit to the size of file Interactive SQL can produce.
- **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.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

output_length option [Interactive SQL]

Function

Controls the length of column values when Interactive SQL exports information to an external file.

Allowed values

Non-negative integer

Default

0 (no truncation)

Description

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 ASCII, HTML, and SQL output formats.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

output_nulls option [Interactive SQL]

Function

Controls the way NULL values are exported.

Allowed values

String

Default

Empty string

Description

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 ASCII, HTML, FIXED, EXCEL, and SQL output formats.

See also

- "SET OPTION statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

truncation_length option [Interactive SQL]

Function

Controls the truncation of wide columns for displays to fit on a screen.

Allowed values

Integer

Default

256

Description

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 fill in the name of any or all of 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, as well as stored procedures and SQL functions.

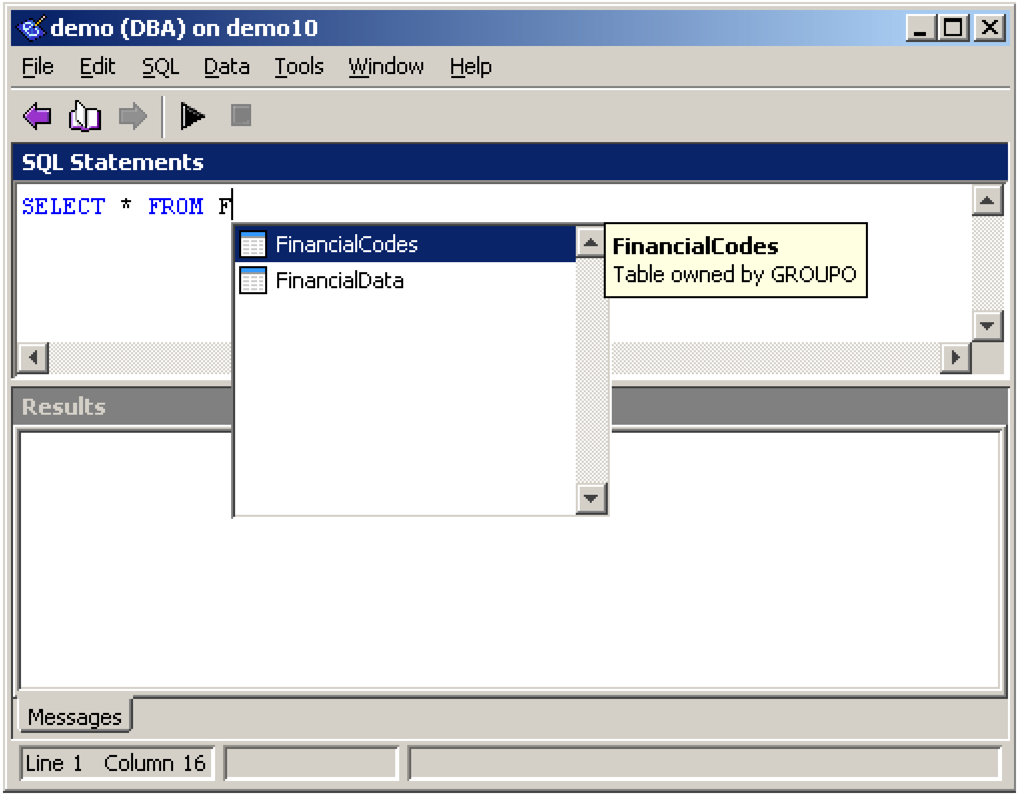
If you open the text completion window after FROM, the list contains only tables and stored procedures.

If you open the text completion window after the second e in WHERE, 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 opens listing the names of database objects that begin with the letter(s) you have typed so far. In the following example, it shows all database objects that begin with the letter F.



```
SELECT * FROM F
```

FinancialCodes
Table owned by GROUPO

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 dialog in Interactive SQL or when you are on 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.
Ctrl+P	Shows only stored procedures and functions in the text completion list.

Key	Description
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.
Escape	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 Quote Identifiers option. See "quoted_identifier option [compatibility]" on page 448.

Using the fast launcher

The fast launcher is designed to reduce the startup time for Sybase Central and Interactive SQL. When the fast launcher is turned on, the fast launcher process (*scjview.exe* for Sybase Central and *dbisqlg.exe* for Interactive SQL) starts when you log in. The fast launcher is only available on Windows.

Configuring the fast launcher

The fast launcher 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 the fast launcher is not used for the amount of time specified in the inactivity timer, it shuts down, which frees up memory for other applications. By default, the inactivity timer is set never to shut down.

♦ To configure the Interactive SQL fast launcher

1. Open Interactive SQL.
2. Choose Tools ▶ Options.

   The Options dialog appears.
3. On the General tab of the Options dialog, click Configure.

   The DBISQL Fast Launcher Configuration dialog appears.
4. Set the fast launcher port and inactivity timer as required.
5. Click OK.

♦ To configure the Sybase Central fast launcher

1. Open Sybase Central.
2. Choose Tools ▶ Options.

   The Options dialog appears.
3. On the General tab of the Options dialog, click Configure.

   The Sybase Central Fast Launcher Configuration dialog appears.
4. Set the fast launcher port and inactivity timer as required.
5. Click OK.

SQL Anywhere Console utility

The SQL Anywhere Console utility provides administration and monitoring facilities for database server connections.

The SQL Anywhere Console utility is available on all supported platforms except Windows CE, AIX, HP-UX, HP-UX Itanium, and Linux Itanium. On these platforms, you can use the connection, database, and server 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 XP/200x, 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 running and using the SQL Anywhere Console utility, see "SQL Anywhere Console utility (dbconsole)" on page 668.

Starting the SQL Anywhere Console utility

♦ To start the SQL Anywhere Console utility (command prompt)

- Execute a `dbconsole` command from a command prompt.

  If you omit the -c option, which specifies the connection parameters for the database, or if you supply insufficient connection parameters, the Connect dialog appears, where you can enter connection information for the database. For information about the supported options, see "SQL Anywhere Console utility (dbconsole)" on page 668.

  The following command starts Interactive SQL and connects to the sample database:

  ```
  dbisql -c "UID=DBA;PWD=sql;DSN=SQL Anywhere 10 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 10.

♦ To start the SQL Anywhere Console utility (Linux Desktop icons)

1. Open the SQL Anywhere 10 folder on your desktop.
2. Double-click DBConsole.

   The SQL Anywhere Console utility opens, and the Connect dialog appears.
3. Enter the connection information for your database in the Connect dialog.

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 278.

- **To start the SQL Anywhere Console utility (Unix command line)**

1. In a terminal session, enter the following command:

   ```
   dbconsole
   ```

   The SQL Anywhere Console utility opens, and the Connect dialog appears.

2. Enter the connection information for your database in the Connect dialog.

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 dialog.

- **To customize the contents of the Connections pane**

1. In the SQL Anywhere Console utility, choose File ▶ Options.

   The Options dialog appears.

2. Click Connection Viewer in the left pane.
3. On the Connection Viewer tab, select the properties you want to appear in the Connections pane.

- **To customize the contents of the Properties pane**

1. In the SQL Anywhere Console utility, choose File ▶ Options.

   The Options dialog appears.

2. Click Property Viewer in the left pane.
3. On the Property Viewer tab, select the database and server properties you want to appear in the Properties pane.

- **To customize the contents of the Messages pane**

1. In the SQL Anywhere Console utility, choose File ▶ Options.

   The Options dialog appears.

2. Click Message Viewer in the left pane.
3. On the Message Viewer tab, select the desired options.

Checking for software updates

You can configure SQL Anywhere to notify you when updates such as EBFs and maintenance releases become available. By default, SQL Anywhere does not check 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. Note that the Update Checker settings for each tool are independent, so you can configure the Update Checker to run on a different frequency for each tool.

♦ To configure the update checker (Sybase Central)

1. Choose Help ► SQL Anywhere 10 ► Configure Update Checker.

   The Check For Updates tab of the Options dialog appears.

2. Configure the Update Checker settings as required.

♦ To configure the update checker (Interactive SQL)

1. Choose Tools ► Options.

   The Options dialog appears.

2. Click Check For Updates.
3. Configure the Update Checker settings as required.

♦ To configure the update checker (SQL Anywhere Console utility)

1. Choose File ► Options.

   The Options dialog appears.

2. Click Check For Updates.
3. Configure the Update Checker settings as required.

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 10 ► Check For Updates.
- **Sybase Central** Choose Help ► SQL Anywhere 10 ► 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
  ```

See also

- “Options dialog: Check for Updates tab” [*SQL Anywhere 10 - Context-Sensitive Help*]

CHAPTER 16

Database Administration Utilities

Contents

About this chapter

SQL Anywhere includes a set of utility programs for backing up databases and performing other database administration tasks. This chapter provides reference information for each of the database administration utilities.

Administration utilities overview

This chapter presents reference information on the programs and database administration utilities that are part of SQL Anywhere. Each of the utilities can be accessed from one or more of Sybase Central, Interactive SQL, or at a command prompt.

☞ For comprehensive documentation on Sybase Central, see the Sybase Central online help. For an introduction to Sybase Central, see "Sybase Central" on page 524.

The administration utilities use a set of registry entries or *.ini* files. These are described in "Registry and INI files" on page 299.

Database file administration statements

A set of SQL statements are available that perform some of the tasks that the administration utilities perform. These statements are listed in "SQL Statements" [*SQL Anywhere Server - SQL Reference*].

Using configuration files

Many of the utilities provided with SQL Anywhere 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 @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. In some cases, conflicting options result in an error.

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 the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file.

☞ For more information about obfuscating the contents of a configuration file, see "File Hiding utility (dbfhide)" on page 608.

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 the default location of *samples-dir*, see "The samples directory" on page 295.

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	dbltm	dbstop	dbxtract
dbdsn	dbinit	dbmlsync	dbsupport	mlsrv10
dbeng10	dblic	dbping	dbsvc	mluser
dberase	dblocate	dbremote	dbunload	qaagent

dbfhide	dblog	dbspawn	dbupgrad	rteng10
dbhist	dblsn	dbsrv10	dbvalid	

Usage

To be treated as a directive, the first non-white 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.

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;ENG=myserver;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
```

The Backup utility

You can use the Backup utility to back up the client-side database files and transaction logs for running databases. The Backup utility creates an image backup, which consists of a copy of the database file and/or the transaction log, each as separate files.

☞ For information about making archive backups (a single file that contains both the database file and the transaction log), see "Making archive backups" on page 775.

You can access the Backup utility in the following ways:

♦ From Sybase Central, using the Create Backup Images wizard.

♦ At a command prompt, using the dbbackup command. This is useful for incorporating backup procedures into batch or command files.

♦ From Interactive SQL, using the BACKUP DATABASE statement.

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.

Running the Backup utility on a running database is equivalent to copying the database files when the database is not running. Thus, you can back up the database while other applications or users are using it.

☞ For a description of suggested backup procedures, see "Backup and Data Recovery" on page 761.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

Create Backup Images wizard

The Create Backup Images wizard helps you create a backup of your database and/or transaction log from Sybase Central.

♦ To back up a database file or a running database (Sybase Central)

1. From the Tools menu, choose SQL Anywhere 10 ► Create Backup Images.

   The Create Backup Images wizard appears.

2. Follow the instructions in the wizard.

☞ For full information on backing up a database from Sybase Central, see "Making image backups" on page 771.

Tip
You can also access the Create Backup Images Database wizard from within Sybase Central using any of the following methods:

- Selecting a database, and choosing Create Backup Images from the File menu.
- Right-clicking a database, and choosing Create Backup Images from the popup menu.

Backup utility (dbbackup)

Syntax

dbbackup [*options*] *target-directory*

Option	Description
@data	Read options from the specified environment variable or configuration file.
-b *block-size*	Specify the maximum block size (in number of pages) used to transfer pages from the database server to dbbackup.
-c *"keyword=value; …"*	Supply database connection parameters.
-d	Back up only the main database file.
-k *checkpoint-log-copy-option*	Control copying of checkpoint log.
-l *filename*	Make a live backup of the transaction log to a file.
-n	Change the naming convention for the backup transaction log.
-o *filename*	Log output messages to a file.
-q	Run in quiet mode—do not display messages.
-r	Rename and start a new transaction log.
-s	Perform an image backup on the server using the BACKUP statement.
-t	Back up only the transaction log.
-x	Delete and restart the transaction log.
-xo	Delete and restart the transaction log without making a backup.
-y	Replace files without confirmation.
target-directory	Specify the directory to which the backup files are copied.

Description

The Backup utility (dbbackup) creates a client-side backup. To perform a server-side backup, use the BACKUP DATABASE statement. See "BACKUP statement" [*SQL Anywhere Server - SQL Reference*].

If neither of the options -d or -t are used, all database files are backed up.

If -s is specified, the backup is created on the server using the BACKUP DATABASE statement. Otherwise, the backup is made on the client computer.

Backup utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

☞ For more information, see "File Hiding utility (dbfhide)" on page 608.

Block size (-b)

Use this option to specify 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.

Connection parameters (-c)

Specify connection parameters. The user ID must have DBA authority or REMOTE DBA authority to connect to the database.

☞ For a description of the connection parameters, see "Connection parameters" on page 206.

For example, the following command backs up the sample database running on the server sample_server, connecting as the DBA user, into the *SQLAnybackup* directory:

```
dbbackup -c "ENG=sample_server;DBN=demo;UID=DBA;PWD=sql" SQLAnybackup
```

Back up main database only (-d)

Back up the main database files only, without backing up the transaction log file, if one exists.

Control copying of checkpoint log (-k)

This 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 cause the backup to be performed on the server, the default setting for -k is auto; otherwise, the default setting is copy.

- ♦ **auto** When you specify 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 specified.
- ♦ **copy** When you specify copy, the backup reads the database files without applying pre-images for any modified pages. The checkpoint log in its entirety, as well as the system dbspace, is copied to the backup

directory. The next time the database is started, the database server automatically recovers the database to its 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** When you specify 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** When you specify 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 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.

Live backup (-l, lowercase L)

This option is provided to enable a secondary system to be brought up rapidly in the event of a server crash. A live backup does not terminate, but 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.

If you specify -l, then you cannot use -s to create an image back up on the server.

☞ For more information about live backups, see "Differences between live backups and transaction log mirrors" on page 780 and "Making a live backup" on page 797.

Change backup transaction log naming convention (-n)

This option is used in conjunction with -r. It 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.

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
```

Log output messages to file (-o)

Write output messages to the named file.

Operate quietly (-q)

Do not display output messages. This option is available only when you run this utility from a command prompt.

Rename and start new transaction log (-r)

This option forces a checkpoint and 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.

Perform an image backup on the server (-s)

This option allows you to create 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 recover option.

Back up the transaction log file only (-t)

This 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).

Delete and restart the transaction log (-x)

Back up the existing transaction log, then delete the original log and start a new transaction log.

Delete and restart the transaction log without a backup (-xo)

Delete the current transaction log and start a new one. This operation does not perform a backup; its purpose is to free up disk space in non-replication environments.

Operate without confirming actions (-y)

Choosing this option 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

The directory to which the backup files are copied. If the directory does not exist, it is created. However, the parent directory must exist.

The Data Source utility

The Data Source utility is a cross-platform alternative to the ODBC Administrator for creating, deleting, describing, and listing SQL Anywhere ODBC data sources.

☞ For information about creating a data source using the ODBC Administrator, see "Working with ODBC data sources" on page 64.

Data Source utility (dbdsn)

Syntax

```
dbdsn [ modifier-options ]
  { -l[ s | u ] [ -qq ]
  | -d[ s | u ] dsn
  | -g[ s | u ] dsn
  | -w[ s | u ] dsn [details-options;...]
  | -cl[ -qq ] }
```

Options

Major option	Description
@data	Read options from the specified environment variable or configuration file.
-l[**s** | **u**] [**-qq**]	List either all SQL Anywhere user or system data sources. By default, user data sources are listed. Using -qq with this option lists the DSNs without any messages or titles. You can only specify [u | s] on Windows.
-d[**s** | **u**] *dsn*	Delete the named SQL Anywhere user or system data source. User data sources is the default. You can only specify [u | s] on Windows.
-g[**s** | **u**] *dsn*	List (get) details about the named SQL Anywhere user or system data source. User data sources is the default. You can only specify [u | s] on Windows.
-w[**s** | **u**] *dsn* [*details-options*]	Create (write) a SQL Anywhere user or system data source definition. User data sources is the default. You can only specify [u | s] on Windows.
-cl[**-qq**]	List available connection parameters. Using -qq with this option lists the available connection parameters without any messages or titles.

Modifier-options	Description
-b	Print the connection string for the data source (brief format).
-cm	Display the data source creation command.

Modifier-options	Description
-dr	Include the DRIVER parameter when displaying DSNs.
-f	Display the location of the system information file (Unix).
-ns	Use environment variable settings and do not search for the system information file (Unix).
-o *filename*	Log output messages to a file.
-pe	Encrypt the password field in the data source.
-q	Run in quiet mode—do not display messages.
-v	Print connection parameters in tabular form (verbose format).
-y	Delete or overwrite data source without confirmation.

Details-options	Description
-c "*keyword***=***value***;…"**	Supply database connection parameters.
-cw	Ensure that the DBF parameter (specified using -c) is an absolute filename. If the value of DBF is not an absolute filename, the Data Source utility prepends the current working directory (CWD).

Description

The Data Source utility is a cross-platform alternative to the ODBC Administrator for creating, deleting, describing, and listing SQL Anywhere ODBC data sources. The utility is useful for batch operations. On Windows operating systems, the data sources are held in the registry.

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 SQLANY10 environment variable.

☞ For more information about the system information file, see “Using ODBC data sources on Unix” on page 67.

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.

The modifier options can occur before or after the major option specification.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

Data Source utility options

When you use the Data Source utility, you can choose from major options, modifier options, and details options.

Major options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

☞ For more information, see "File Hiding utility (dbfhide)" on page 608.

List available connection parameters (-cl)

This convenience option lists the connection parameters supported by the dbdsn utility.

☞ For descriptions of the SQL Anywhere connection parameters supported by the Data Source utility (dbdsn), see "Connection parameters" on page 206.

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.
DescribeCursor	This parameter lets you specify how often you want a cursor to be re-described 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.

Name	**Description**
Description	This parameter allows you to provide a description of the ODBC data source.
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 10**. 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 SQLANY10 environment variable.
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.
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.

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. Few inconsistences are possible and performance is the same as isolation level 0 with respect to contention. ♦ **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. Few inconsistences are possible and performance is the same as isolation level 0 with respect to contention. ♦ **readonly-statement-snapshot** This is also called the isolation level. 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. Few inconsistences are possible and performance is the same as isolation level 0 with respect to contention. ☞ For more information, see "Choosing isolation levels" [*SQL Anywhere Server - SQL Usage*].
KeysInSQLStatistics	Specify 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 Visual Basic and Access) assume that primary and foreign keys are returned by SQLStatistics.
LazyAutocommit	Setting this parameter to YES delays the commit operation until a statement closes.

Name	Description
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.
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.

Delete the named data source (-d)

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**.

Include the Driver parameter when displaying data sources (-dr)

Includes the Driver parameter when displaying DSNs. This is particularly useful when using the -cm option to recreate DSNs because it allows the current version of dbdsn to create DSNs that reference a different version of the ODBC driver.

For example, suppose you used the following command to create a version 9.0 DSN:

```
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 DSN 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.

List (get) details of the named data source (-g)

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**.

List defined data sources (-l)
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**.

Create (write) a data source definition (-w)
Creates a new data source, or overwrites one if one of the same name exists. 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**.

Modifier options

Print connection string for the data source (-b) Format the output of the list as a single line connection string.

Display the data source creation command (-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"
```

Display the location of the system information file (-f)
When this option is specified, dbdsn displays the name of the system file that is being used. This option is only available on Unix.

Use environment variable settings to locate the system information file (-ns)
You can use the -ns option when creating a data source on Unix to specify 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.

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 information about how the system information file is located, see "Using ODBC data sources on Unix" on page 67.

Log output messages to file (-o)
Write output messages to the named file.

Encrypt the password in the data source (-pe)

When this option is specified and a PWD entry is included in the DSN, dbdsn encrypts the password in the PWD entry, and replaces the PWD entry with an ENP entry containing the encrypted password.

Do not display messages (-q)

Suppress output to the Server Messages window. If you specify -q when deleting or modifying a data source, you must also specify -y.

Do not print messages or titles (-qq)

Suppress output to the Server Messages window. If the -o option is specified, titles are also suppressed in the output. This option can only be used with the -l and the -cl options.

Print connection parameters in tabular form (-v)

Format the output of the list over several lines, as a table.

Delete or overwrite data source without confirmation (-y)

Automatically delete or overwrite 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

Connection parameters (-c)

Specify connection parameters as a connection string.

☞ For more information, see “Connection parameters” on page 206.

Absolute filenames (-cw)

Ensure 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.

See also

♦ “Working with ODBC data sources” on page 64

♦ “Using ODBC data sources on Unix” on page 67

Examples

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;LINKS=TCPIP;ENG=myserver"
```

or, with a different option order,

```
dbdsn -w newdsn -c "UID=DBA;PWD=sql;LINKS=TCPIP;ENG=myserver" -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 user data source MyDSN:

```
dbdsn -g MyDSN
```

Report the connection string for 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;ENG=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 DSNs (without messages or titles):

```
dbdsn -l -qq -o dsninfo.txt
```

List all connection parameter names (without messages or titles):

```
dbdsn -cl -qq -o dsninfo.txt
```

Specify an absolute file name. When the DSN is created, it will contain *DBF=c:\SQLAnywhere10\my.db*.

```
c:\SQLAnywhere10> dbdsn -w testdsn -cw -c UID=DBA;PWD=sql;ENG=SQLAny;DBF=my.db
```

Generate the command to create the SQL Anywhere 10 Demo data source and output it to a file called *restoredsn.bat*:

```
dbdsn -cm -g "SQL Anywhere 10 Demo" > restoredsn.bat
```

The *restoredsn.bat* file contains the following:

```
dbdsn -y -wu "SQL Anywhere 10 Demo" -c "UID=DBA;PWD=sql;
DBF='C:\Documents and Settings\All Users\Documents\SQL Anywhere 10\Samples
\demo.db';
ENG=demo10;START='C:\Program Files\SQL Anywhere 10\win32\dbeng10.exe';
ASTOP=yes;Description='SQL Anywhere 10 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
```

The Erase utility

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.

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 to be erased must not be running when this utility is used.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

You can access the Erase utility in the following ways:

- From Sybase Central, using the Erase Database wizard.
- At a command prompt, using the dberase command. This is useful for incorporating into batch or command files.

☞ For more information, see the "DROP statement" [*SQL Anywhere Server - SQL Reference*].

Erase Database wizard

The Erase Database wizard helps you delete all dbspaces and transaction logs associated with a database.

♦ To erase a database file (Sybase Central)

1. From the Tools menu, choose SQL Anywhere 10 ► Erase Database.

   The Erase Database wizard appears.

2. Follow the instructions in the wizard.

☞ For full information on erasing a database from Sybase Central, see "Erasing a database" [*SQL Anywhere Server - SQL Usage*].

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

- Selecting a database server, and choosing Erase Database from the File menu.
- Right-clicking a server, and choosing Erase Database from the popup menu.

Erase utility (dberase)

Syntax

dberase [*options*] *database-file*

Option	Description
@data	Read options from the specified environment variable or configuration file.
-ek *key*	Specify encryption key.
-ep	Prompt for encryption key.
-o *filename*	Log output messages to a file.
-q	Run in quiet mode—do not display messages.
-y	Erase files without confirmation.

Description

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.

Note
The Erase 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.

Erase utility options

Specify environment variable or configuration file (@*data*)
Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

☞ For more information, see "File Hiding utility (dbfhide)" on page 608.

Specify encryption key (-ek)

This option allows you to specify 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.

Prompt for encryption key (-ep)
This option allows you to specify that you want to be prompted for the encryption key. This option causes a dialog box 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.

Log output messages to file (-o)
Write output messages to the named file.

Operate quietly (-q)
Do not display output messages. If you specify this option, you must also specify -y or the operation will fail.

Operate without confirming actions (-y)
Choosing this option automatically deletes each file without prompting you for confirmation. If you specify -q, you must also specify -y or the operation will fail.

The File Hiding utility

With the File Hiding utility you can add simple encryption to configuration files and initialization files to hide the contents of each file.

File Hiding utility (dbfhide)

Syntax

dbfhide *original-configuration-file encrypted-configuration-file*

Option	Description
original-configuration-file	Specify the name of the original file.
encrypted-configuration-file	Specify a name for the new obfuscated file.

Description

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, as well as 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.

Hiding the contents of .ini files

In many cases, 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 need to 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 .ini files

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.

This utility does *not* accept the ***@data*** parameter to read in options from a configuration file.

Examples

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 the default location of *samples-dir*, see "The samples directory" on page 295.

Name the file *sample.txt*. If you wanted to start the database using this configuration file, your command line would be:

```
dbeng10 @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.

```
dbsrv10 @encrypted_sample.txt
```

☞ For more information about encryption, see "Keeping Your Data Secure" on page 855.

☞ For more information about using configuration files, see "Using configuration files to store server startup options" on page 18.

The Histogram utility

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 Excel 97 or later installed.

Histogram utility (dbhist)

Syntax

dbhist [*options*] **-t** *table-name* [*excel-output-filename*]

Options

Option	Description
@data	Read options from the specified environment variable or configuration file.
-c *options*	Supply the connection string.
-n *colname*	Specify the column to associate with the histogram.
-t *table-name*	Specify the name of the table or materialized view for which to generate histograms.
-u *owner*	Specify the owner of the table or materialized view.
excel-output-name	Specify the name of the generated Excel file.

Description

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 Excel 97 or later installed.

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 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 an 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).

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

☞ For more information about the sa_get_histogram stored procedure, see "sa_get_histogram system procedure" [*SQL Anywhere Server - SQL Reference*].

Histogram utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

☞ For more information, see "File Hiding utility (dbfhide)" on page 608.

Specify a connection string (-c)

Specify connection parameters.

☞ For a description of the connection parameters, see "Connection parameters" on page 206.

Specify a column (-n)

Specify 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.

Specify a table or materialized view name (-t)

Specify the name of the table or materialized view for which to generate histograms.

Specify an owner (-u)

Specify the owner of the table.

excel-output-name The name of the generated Excel file. If no name is specified, Excel prompts you to enter one with a Save As dialog.

Example

Assuming that a histogram has been created for the column, the following command (entered all on the same line) generates an Excel chart for the column ProductID in the table SalesOrderItems for database *demo.db*, and saves it as *histgram.xls*.

```
dbhist -c "UID=DBA;PWD=sql;DBF=samples-dir\demo.db" -n ProductID -t
SalesOrderItems histgram.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 Excel prompts you to enter one.

```
dbhist -t SalesOrders -c "UID=DBA;PWD=sql"
```

☞ For information about the default location of *samples-dir*, see "The samples directory" on page 295.

The Information utility

The Information utility displays information about a database.

Information utility (dbinfo)

Syntax

dbinfo [*options*]

Option	Description
@data	Read options from the specified environment variable or configuration file.
-c *"keyword=value; …"*	Database connection parameters.
-o *filename*	Log output messages to a file.
-q	Run in quiet mode—do not display messages.
-u	Output page usage statistics.

Description

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.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

Information utility options

Specify environment variable or configuration file (@*data*)
Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Connection parameters (-c)
Specify connection parameters.

☞ For a description of the connection parameters, see "Connection parameters" on page 206.

Any valid user ID can run the Information utility, but to obtain page usage statistics you need DBA authority.

Log output messages to file (-o)
Write output messages to the named file.

Operate quietly (-q)
Do not display output messages.

Page usage statistics (-u)
Display information about the usage and size of all tables, including system and user-defined tables, as well as 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.

The Initialization utility

With the Initialization utility, you can initialize (create) a database.

Create Database wizard

The Create Database wizard helps you create a new database from Sybase Central.

♦ To create a database (Sybase Central)

1. From the Tools menu, choose SQL Anywhere 10 ▶ Create Database.

   The Create Database wizard appears.

2. Follow the instructions in the wizard.

Tip
You can also access the Create Database wizard from within Sybase Central using any of the following methods:

- Selecting a server, and choosing Create Database from the File menu.
- Right-clicking a server, and choosing Create Database from the popup menu.

☞ For full information on creating a database in Sybase Central, see "Creating a database" [*SQL Anywhere Server - SQL Usage*].

Initialization utility (dbinit)

Syntax

dbinit [*options*] *new-database-file*

Option	Description
@data	Read options from the specified environment variable or configuration file.
-a	Use accent sensitivity on all UCA string comparisons.
-af	Use accent sensitivity (French rules) on all UCA string comparisons.
-b	Use blank padding of strings for comparisons and fetching.
-c	Use case sensitivity for all string comparisons.
-dba [*DBA-user*][**,***pwd*]	Specify a name and/or password for the default DBA user.

Option	Description
-dbs *size*[**k** | **m** | **g** | **p**]	Specify the initial size of the database.
-e	Encrypt the database using simple encryption.
-ea *algorithm*	Encrypt with the specified strong encryption algorithm: you can choose AES or AES_FIPS.
-ek *key*	Specify encryption key for strong encryption.
-ep	Prompt for encryption key for strong encryption.
-et	Enable table encryption in the database.
-i	Do not install Sybase jConnect catalog support.
-k	Omit Watcom SQL compatibility views SYS.SYSCOLUMNS and SYS.SYSINDEXES.
-l	List the recommended collation sequences.
-m *file-name*	Use a transaction log mirror (default is no mirror).
-n	Do not use a transaction log.
-o *filename*	Log output messages to a file.
-p *page-size*	Set page size.
-q	Run in quiet mode—do not display messages.
-s	Use checksums when writing pages to disk.
-t *log-name*	Specify the transaction log file name (the default is the database name with *.log* extension).
-z *coll*	Specify the collation sequence for the CHAR data type (defaults to server platform's codepage and ordering).
-ze *encoding*	Specify the character set encoding for the CHAR data type.
-zn *coll*	Specify the collation sequence for the NCHAR data type.

Description

A number of 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
- 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
```

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.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

You can create a database in the following ways:

- From Sybase Central, using the Create Database wizard.
- From Interactive SQL, using the CREATE DATABASE statement.
- At a command prompt, using the dbinit command. This is useful for incorporating into batch or command files.

☞ For more information about creating a database, see "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

Initialization utility options

Specify environment variable or configuration file (@*data*)
Use this option to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Accent sensitivity for UCA string comparisons (-a)
If the UCA (Unicode Collation Algorithm) is used for either CHAR or NCHAR data types (see -z and -zn), then specifying -a causes string comparisons to respect accent differences between letters (for example, e is less than é). 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*.

☞ For more information, see "Unicode Collation Algorithm (UCA)" on page 318.

Accent sensitivity, French rules, for UCA string comparisons (-af)
If the UCA is used for either CHAR or NCHAR data types (see -z and -zn below), then specifying -af causes string comparisons to respect accent differences between letters (for example, e is less than é). 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 318.

Blank padding (-b)

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. Hence 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. Whether or not the application receives a string truncation warning on such an assignment is controlled by the ansi_blanks connection option. See “ansi_blanks option [compatibility]” on page 390.

Case sensitivity for all string comparisons (-c)

For databases created with this option, all values are considered to be case sensitive in comparisons and string operations. Identifiers in the database are case insensitive, even in case sensitive databases.

This option is provided for compatibility with the ISO/ANSI SQL standard. The default is that all comparisons are case insensitive.

Specify DBA user and password (-dba)

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.

Set initial database size (-dbs)

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**, or **g** to specify units of kilobytes, megabytes, or gigabytes, respectively. The unit **p** specifies pages.

Encrypt the database using simple encryption (-e)

Simple encryption makes it more difficult for someone to decipher the data in your database using a disk utility to look at the file. File compression utilities cannot compress encrypted database files as much as unencrypted ones.

This 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. For greater security, you can use strong encryption by supplying the -ea and -ek options.

Specify encryption algorithm (-ea)

This option allows you to choose a strong encryption algorithm for encryption. You can choose either AES (the default) or AES_FIPS for the FIPS-approved algorithm. AES_FIPS uses a separate library. Algorithm names are case insensitive. If you specify the -ea option, you must also specify -ep or -ek.

The following command creates a strongly encrypted database and specifies the encryption key and algorithm.

```
dbinit -ek "0kZ2o56AK#" -ea AES_FIPS "myencrypteddb.db"
```

☞ For more information, see “Strong encryption” on page 873.

Separately licensed component required

ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.

See “Separately licensed components” [*SQL Anywhere 10 - Introduction*].

Specify encryption key (-ek)

This option allows you to create a strongly encrypted database by specifying an encryption key directly in the command. The algorithm used to encrypt the database is AES or AES_FIPS as specified by the -ea option. If you specify the -ek option without specifying -ea, the AES algorithm is used.

When specified with -et, the database is not encrypted. Instead, table encryption is enabled.

Caution

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.

Prompt for encryption key (-ep)

This option allows you to specify that you want to create a strongly encrypted database by inputting the encryption key in a dialog box. 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 873.

Enable table encryption (-et)

This option enables table encryption for the database, allowing you to create encrypted tables instead of encrypting the entire database. If you specify the -et option with -ek or -ep, the AES algorithm is used. If you specify the -ea option, you must also specify -ek or -ep. 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 tables" on page 881.

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
```

Do not install Sybase jConnect catalog support (-i)

If you want to use the Sybase jConnect JDBC driver to access system catalog information, you need to install jConnect catalog support (it is installed by default). Use this option if you want to exclude the jConnect system objects. You can still use JDBC, as long as you do not access system information. If you want, you can add Sybase 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*].

Omit Watcom SQL compatibility views (-k)

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 will conflict with the Sybase Adaptive Server Enterprise compatibility views dbo.syscolumns and dbo.sysindexes.

List the available collation sequences (-l)

When you specify this option, dbinit 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.

Use a transaction log mirror (-m)

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 mirrored transaction log.

Do not use a transaction log (-n)

Creating a database without a transaction log saves disk space. The transaction log is required for data replication and provides extra security for database information in case of media failure or system failure. Databases that do not use transaction logs typically run slower than databases that use transaction logs.

Log output messages to file (-o)

Write output messages to the named file.

Page size (-p)

The page size for a database can be (in bytes) 2048, 4096, 8192, 16384, or 32768, with 4096 being the default.

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 tradeoffs of memory and disk space with its performance characteristics. It is particularly important to pick the correct (and reasonable) page size if a large number of databases are going to be started on the same server.

☞ For more information, see:

- "Use an appropriate page size" [*SQL Anywhere Server - SQL Usage*]
- "Table and page sizes" [*SQL Anywhere Server - SQL Usage*]

Operate quietly (-q)

Do not display output messages.

Use checksums (-s)

Checksums are used to determine whether a database page has been modified on disk. When you create a database with 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 whether -s is specified.

Creating databases that will be deployed to Windows CE

If you are creating a database that will be deployed to Windows CE, you should enable checksums. This helps to provide early detection if the database file becomes corrupt.

Set the transaction log file name (-t)

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 (see "The transaction log" on page 766), 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*.

Collation sequence for CHAR data types (-z)

The collation sequence is used for sorting and comparison of character data types (CHAR, VARCHAR, LONG VARCHAR). The collation provides character comparison and ordering information for the encoding (character set) being used.

If the collation is not specified, SQL Anywhere chooses a collation based on the operating system language and character set.

It is important to choose your collation carefully. It cannot be changed after the database has been created without unloading and reloading the database. See "Choosing collations" on page 320.

Encoding for CHAR data types (-ze)

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 UCA (Unicode Collation Algorithm), 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.

Collation sequence for NCHAR data types (-zn)
The collation sequence used for sorting and comparison of national character data types (NCHAR, NVARCHAR, LONG NVARCHAR) is specified using -zn. 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.

☞ For more information, see “Choosing collations” on page 320.

The Interactive SQL utility

Interactive SQL provides an interactive environment for database browsing and for sending SQL statements to the database server.

You can start Interactive SQL in the following ways:

- from Sybase Central, using the Open Interactive SQL menu item.
- from the Start menu by choosing Start ▶ Programs ▶ SQL Anywhere 10 ▶ Interactive SQL.
- using the dbisql command.

Interactive SQL utility (dbisql)

Syntax

dbisql [*options*] [*dbisql-command* | *command-file*]

Option	Description
@data	Read options from the specified environment variable or configuration file.
-c *"keyword=value; ..."*	Supply database connection parameters.
-codepage *codepage*	Specify a codepage to use when reading or writing files.
-d *delimiter*	Use the given string as the command delimiter.
-d1	Print statements as they are executed [command-prompt mode only].
-datasource *DSN-name*	Specify an ODBC data source to connect to.
-f *filename*	Open (without running) the file called *filename*.
-host *hostname*	Specify the *hostname* or IP address of the computer running a database server.
-nogui	Run in command-prompt mode.
-onerror { **continue** | **exit** }	Override the on_error option setting.
-port *port-number*	Look on the specified port number for the database server.
-q	Run in quiet mode—do not display messages or windows. (Note that this does not suppress error messages.)
-x	Syntax check only—no commands executed.
-ul	Connect to UltraLite databases by default.

Description

Interactive SQL allows you to type SQL commands or run command files. It also provides feedback about the number of rows affected, the time required for each command, the execution plan of queries, and any error messages.

You can connect to both SQL Anywhere and UltraLite databases.

The dbisql utility is supported on Windows XP/200x, Solaris, Linux, and Mac OS X.

If *dbisql-command* is specified, Interactive SQL executes the command. You can also specify a command file name. If no *dbisql-command* or *command-file* argument is specified, Interactive SQL enters interactive mode, where you can type a command into a command window.

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).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

Interactive SQL utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Connection parameters (-c)

Specify connection parameters. See "Connection parameters" on page 206 for a description of the connection parameters. If Interactive SQL cannot connect, you are presented with a dialog where you can enter the connection parameters.

Codepage (-codepage)

Specify the codepage to use when reading or writing files. The default code page is the default code page for the platform you are running on.

For example, on an English Windows XP computer, windowed programs use the 1252 (ANSI) code page. If you want Interactive SQL to read files created using the 297 (IBM France) code page, specify the following option.

```
-codepage 297
```

☞ For more information on recommended code pages, see "Recommended character sets and collations" on page 330.

The default codepage for Interactive SQL can also be set using the default_isql_encoding option. You can also select the codepage when issuing an INPUT, OUTPUT, or READ statement by specifying the ENCODING clause.

☞ For more information, see

- “default_isql_encoding option [Interactive SQL]” on page 563
- “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*]

Command delimiter (-d)

Specify 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 562.

Echo statements (-d1)

(The final character is a number 1, not a lowercase L). Interactive SQL 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.

Data source (-datasource)

Specify an ODBC data source to connect to.

Open file using Interactive SQL (-f)

Open (but do not run) the file called *filename*. 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 to the console 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.

Host (-host)

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.

Run in command-prompt mode (-nogui)

Run Interactive SQL in a command-prompt mode, 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.

☞ For information about possible exit codes, see “Software component exit codes” [*SQL Anywhere Server - Programming*].

Override the on_error option setting (-onerror)

Controls what happens if an error is encountered while reading statements from a command file. This option overrides the on_error setting. It is useful when using Interactive SQL in batch operations. See “on_error option [Interactive SQL]” on page 571.

Database server port (-port)

Specify the port number on which the database server is running. The default port number for SQL Anywhere is 2638.

Operate quietly (-q)

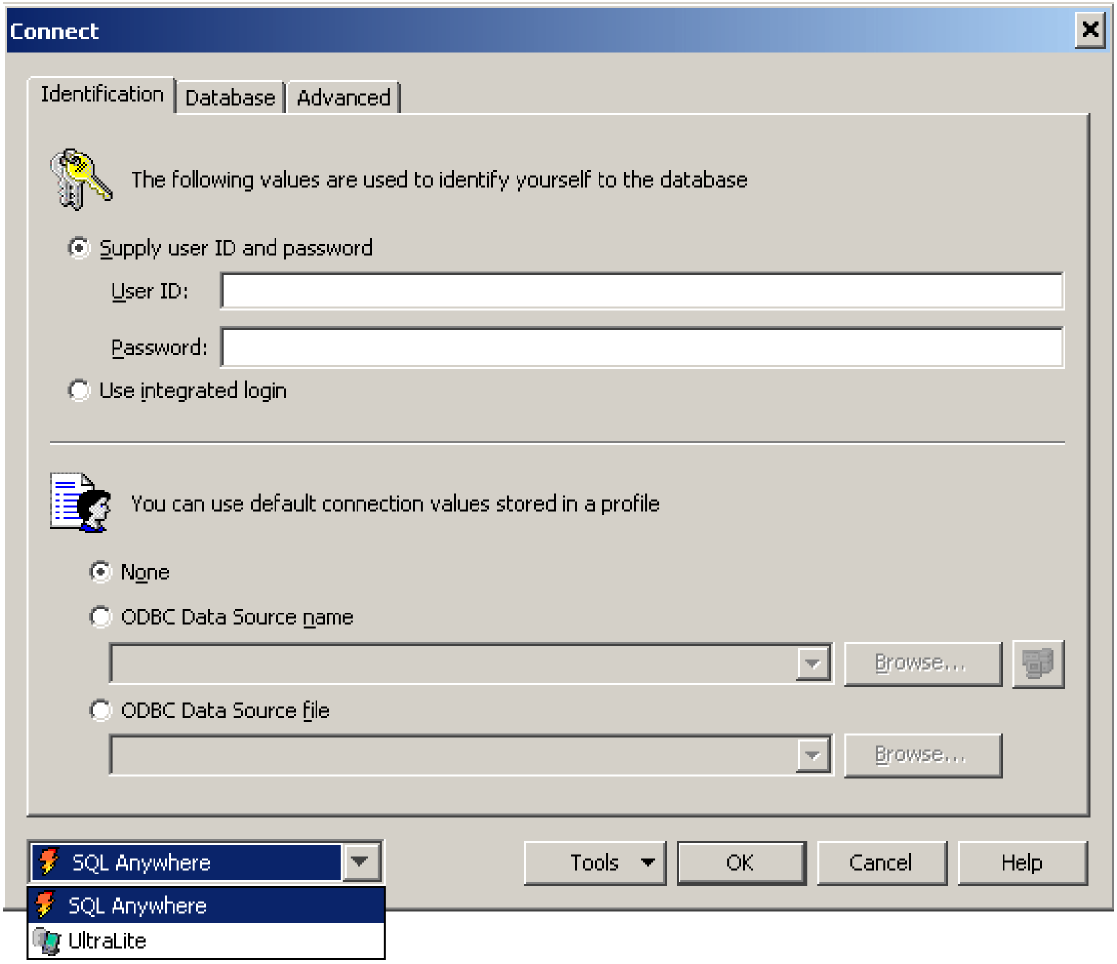
Do not display output messages. This is useful only if you start Interactive SQL with a command or command file. Specifying this option does not suppress error messages, but it does suppress the following:

- warnings and other non-fatal messages
- the printing of result sets

Syntax check only (-x)
Scan commands but do 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*].

Connect to UltraLite databases by default (-ul)
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 dropdown list on the Connect dialog.



The following values are used to identify yourself to the database

● Supply user ID and password

○ Use integrated login

You can use default connection values stored in a profile

● None

○ ODBC Data Source name

○ ODBC Data Source file

SQL Anywhere

☞ For more information about connecting to UltraLite databases from Interactive SQL, see "Interactive SQL utility (dbisql)" [*UltraLite - Database Management and Reference*].

Examples

The following command, entered at a command prompt, 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 terminates.

```
dbisql -c "UID=DBA;PWD=sql" -onerror exit mycom.sql
```

The following command, when entered on a single line at a command prompt, adds a user to the current default database:

```
dbisql -c "UID=DBA;PWD=sql" GRANT CONNECT TO joe IDENTIFIED passwd
```

Interactive SQL utility (dbisqlc)

Syntax

dbisqlc [*options*] [*dbisqlc-command* | *command-file*]

Option	Description
-c *"keyword=value; …"*	Supply database connection parameters.
-d *delimiter*	Use the given string as the command delimiter.
-q	Run in quiet mode—do not display messages or windows. (Note that this does not suppress error messages.)
-x	Syntax check only—no commands executed.

Description

Note
It is recommended that you use the Interactive SQL utility (accessed by using the dbisql command or by choosing Start ▶ Programs ▶ SQL Anywhere 10 ▶ Interactive SQL) where possible because the dbisqlc utility does not support all the features that Interactive SQL does, and it does not support all of the features available in SQL Anywhere 8, 9, and 10 databases and database servers.

The dbisqlc utility allows you to type SQL commands or run command files.

The dbisqlc utility is supported on Windows XP/200x, NetWare, Mac OS X, and Unix.

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.

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).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

Interactive SQL utility options

Connection parameters (-c)

Specify connection parameters. See "Connection parameters" on page 206 for a description of the connection parameters. If Interactive SQL cannot connect, you are presented with a dialog where you can enter the connection parameters.

Command delimiter (-d)

Specify 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.

Operate quietly (-q)

Do not display 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

Syntax check only (-x)

Scan commands but do 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*].

Examples

The following command, entered at a command prompt, 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 terminates.

```
dbisqlc -c "UID=DBA;PWD=sql" mycom.sql
```

The following command, when entered on a single line at a command prompt, adds a user to the current default database:

```
dbisqlc -c "UID=DBA;PWD=sql" GRANT CONNECT TO joe IDENTIFIED passwd
```

The Language Selection utility

The Language Selection utility reports and changes the registry settings that control the languages used by SQL Anywhere and Sybase Central.

Language Selection utility (dblang)

Syntax

dblang [*options*] *language-code*

Option	Description
-q	Run in quiet mode—do not print messages.

Language code	Language
EN	English
DE	German
ES	Spanish
FR	French
IT	Italian
JA	Japanese
KO	Korean
LT	Lithuanian
PL	Polish
PT	Portuguese
RU	Russian
TW	Traditional Chinese
UK	Ukrainian
ZH	Simplified Chinese

Description

The utility is installed only when the International Resources Deployment Kit (IRDK) is selected during installation.

Running the dblang utility without a language code reports the current settings. These settings are as follows:

- **SQL Anywhere** A key from HKEY_LOCAL_MACHINE that holds a two-letter language code from the table above.

  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 *dblgXX10.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** A key from HKEY_LOCAL_MACHINE that holds a two-letter language code from the table above.

  This setting controls the resources used to display user interface elements for Sybase Central and Interactive SQL. You must have purchased the appropriate localized version of SQL Anywhere for this setting to take effect.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, 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.

Language Selection utility options

Operate quietly (-q)
Do not display output messages.

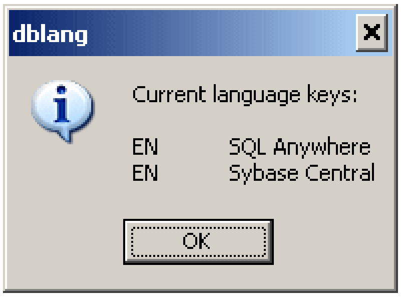
See also

- "SALANG environment variable" on page 284

Examples

If you selected the International Resources Deployment Kit (IRDK) during installation, the following command displays a dialog box containing the current settings:

```
dblang
```



Current language keys:

```
EN        SQL Anywhere
EN        Sybase Central
```

The following command changes the settings to French, and displays a dialog containing the previous and new settings:

```
dblang FR
```

The Log Transfer Manager

The Log Transfer Manager (LTM) is also known as a **replication agent**.

The LTM is required for any SQL Anywhere database that participates in a Replication Server installation as a primary site.

Log Transfer Manager utility (dbltm)

Syntax

dbltm [*options*]

Options

Option	Description
@data	Read options from the specified environment variable or configuration file.
-A	Do not filter updates.
-C *config_file*	Use the named configuration file.
-I *interface_file*	Use the named interface file.
-M	Run in recovery mode.
-S *LTM_name*	Specify an LTM name.
-dl	Display log messages on screen.
-ek *key*	Specify encryption key.
-ep	Prompt for encryption key.
-o *filename*	Log output messages to a file.
-os *size*	Specify the maximum size of output file.
-ot *file*	Truncate file, and log output messages to file.
-q	Run in minimized window.
-s	Show Log Transfer Language (LTL) commands.
-ud	Run as a daemon [Unix].
-ux	Open the console window [Solaris and Linux].
-v	Run in verbose mode.

Description

The SQL Anywhere LTM reads a database transaction log and sends committed changes to Replication Server. The LTM is not required at replicate sites.

The LTM sends committed changes to Replication Server in a language named Log Transfer Language (LTL).

By default, the LTM uses a log file named *DBLTM.LOG* to hold status and other messages. You can use options to change the name of this file and to change the volume and type of messages that are sent to it.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see “Software component exit codes” [*SQL Anywhere Server - Programming*].

Log Transfer Manager utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see “Using configuration files” on page 587.

Do not filter updates (-A)

Do not filter updates. By default, all changes made by the maintenance user are not replicated. If the -A option is set, these changes are replicated. This may be useful in non-hierarchical Replication Server installations, where a database acts as both a replicate site and as a primary site.

Use configuration file (-C)

Use the configuration file *config_file* to determine the LTM settings. The default configuration file is *dbltm.cfg*.

☞ For a description of the configuration file, see “The LTM configuration file” on page 633.

Display Log Transfer Language messages (-dl)

Display all messages in the LTM window or at a command prompt, as well as in the log file (if specified).

Specify encryption key (-ek)

This option allows you to specify 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, including offline transaction logs. For strongly encrypted databases, you must specify either -ek or -ep, but not both. The command fails if you do not specify a key for a strongly encrypted database.

Prompt for encryption key (-ep)

This option allows you to specify that you want to be prompted for the encryption key. This option causes a dialog box 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 a key for a strongly encrypted database.

Use interface file (-I)

(Uppercase I.) Use the named interfaces file. The interfaces file is the file created by DSEDIT and holds the connection information for Open Servers. The default interfaces file is *SQL.ini* in the *ini* subdirectory of your *Sybase* directory.

Recover mode (-M)
This is used to initiate recovery actions. The LTM starts reading logs from the earliest available position. If the offline directory is specified in the configuration file, the LTM reads from the oldest offline log file.

Log output messages to file (-o)
Use a log file different from the default (*dbltm.log*). Write output messages from log transfer operations to this file.

Limit size of output file (-os)
Specify the maximum size of the output file, in bytes. The minimum value is 10000 (ten thousand). If the log file grows to the point where it would exceed this limit, it is renamed to *yymmddxx.ltm*. The value of *xx* in *yymmddxx.ltm* is incremented for each file created on a given day.

Truncate log file and use named log file (-ot)
Use a log file different from the default (*dbltm.log*), and truncate the log file (all existing content is deleted) when the LTM starts. Output messages from log transfer operations are sent to this file for later review.

Quiet mode (-q)
Minimize the window when the LTM is started.

Specify LTM name (-S) Provides the server name for this LTM. The default LTM name is DBLTM_LTM. The LTM name must correspond to the Open Server name for the LTM that was entered in DSEDIT.

Log all LTL commands (-s)
Log all LTL commands that are generated by the LTM. This should be used only to diagnose problems, and is not recommended in a production environment. It carries a significant performance penalty.

Run as a daemon (-ud)
You can run the LTM as a daemon on Unix operating systems. If you run in this manner, output is logged to the log file.

Open the console (-ux)
When -ux is specified, dbltm must be able to find a usable display. If it cannot find one, for example because the DISPLAY environment variable is not set or because the X Windows Server is not running, dbltm fails to start. On Windows, the console appears automatically.

Operate in verbose mode (-v)
Displays messages, other than LTL messages, for debugging purposes.

The LTM configuration file

The SQL Anywhere and Adaptive Server Enterprise LTM configuration files are very similar. This section describes the entries in the SQL Anywhere LTM configuration file, and the differences from the Adaptive Server Enterprise LTM configuration file.

The configuration file that an LTM uses is specified using the -C option.

LTM configuration file parameters

The following table describes each of the configuration parameters that the LTM recognizes. Options that are used by the Adaptive Server Enterprise LTM but not by the SQL Anywhere LTM are included in this list, and marked as either ignored (in which case they may be present in the configuration file, but have no effect) or as unsupported (in which case they will cause an error if present in the configuration file).

Parameter	Description
APC_pw	The password for the APC_user login name. This entry is present only in SQL Anywhere LTM configuration files.
APC_user	A user ID that is used when executing asynchronous procedures at the primary site. This user ID must have permissions appropriate for all asynchronous procedures at the primary site. This entry is present only in SQL Anywhere LTM configuration files.
backup_only	By default, this is **off**. If set to **on**, the LTM will replicate only backed-up transactions.
batch_ltl_cmds	Set to **on** (the default) to use batch mode. Batch mode can increase overall throughput, but may lead to longer response times.
batch_ltl_sz	The number of commands that are saved in the buffer before being sent to Replication Server, when **batch_ltl_cmds** is **on**. The default is 200.
batch_ltl_mem	The amount of memory that the buffer can use before its contents are sent to Replication Server, when **batch_ltl_cmds** is **on**. The default is 256 KB.
Continuous	By default, this is on. When set to off, the LTM automatically shuts down as soon as all committed data has been replicated.
LTM_admin_pw	The password for the LTM_admin_user login name.
LTM_admin_user	The system administrator LTM login name that is used to log in to the LTM. This parameter is required so that the LTM can check whether a user logging on to the LTM to shut it down has the correct login name.
LTM_charset	The Open Client/Open Server character set for the LTM to use.
LTM_language	The Open Client/Open Server language for the LTM to use.
LTM_sortorder	The Open Client/Open Server sort order for the LTM to use to compare user names. You can specify any Adaptive Server Enterprise-supported sort order that is compatible with the LTM's character set. All sort orders in your replication system should be the same. The default sort order is a binary sort.
maint_cmds_to_skip	[ignored]

Parameter	Description
qualify_table_owners	Set to **on** for the LTM to send LTLs with table names and columns names, as well as table owners to Replication Server. The setting applies to all replicating tables, and the create replication definition statements must match this setting. The default is **off**.
rep_func	Set to **on** to use asynchronous procedure calls (APCs). The default is **off**.
Retry	The number of seconds to wait before retrying a failed connection to a SQL Anywhere database server or Replication Server. The default is 10 seconds.
RS	The name of the Replication Server to which the LTM is transferring the log.
RS_pw	The password for the RS_user login name.
RS_source_db	The name of the database whose log the LTM transfers to the Replication Server. This name must match the name of the database as defined within the Replication Server connection definitions. Most configurations use the same setting for both RS_Source_db and SQL_database configuration options.
RS_source_ds	The name of the server whose log the LTM transfers to the Replication Server. This name must match the name of the server as defined within the Replication Server connection definitions. Most configurations use the same setting for both RS_Source_ds and SQL_server configuration options.
RS_user	A login name for the LTM to use to log in to the Replication Server. The login name must have been granted **connect source** permission in the Replication Server.
scan_retry	The number of seconds that the LTM waits between scans of the transaction log. This parameter is somewhat different in meaning to that of the Adaptive Server Enterprise LTM. The SQL Anywhere server does not *wake up* and scan the log when records arrive in the log. For this reason, you may want to set the *scan_retry* value to a smaller number then that for an Adaptive Server Enterprise LTM.
skip_ltl_cmd_err	This parameter tells Replication Agent to continue or shut down when LTL command errors occur. When skip_ltl_cmd_err=on is specified, Replication Agent displays the LTL commands that caused the erros and then skips the LTLs and continues replication. When this parameter is set to off, Replication Agent displays the LTL commands that caused the errors and then shuts down. By default, this parameter is set to off.
SQL_database	The primary site database name on the server SQL_server to which the LTM connects. For Adaptive Server Enterprise during recovery, this is the temporary database whose logs the LTM will transfer to Replication Server. The SQL Anywhere LTM uses the SQL_log_files parameter to locate offline transaction logs.

Parameter	Description
SQL_log_files	A directory that holds off-line transaction logs. The directory must exist when the LTM starts up. This entry is present only in SQL Anywhere LTM configuration files.
SQL_pw	The password for the SQL_user user ID.
SQL_server	The name of the primary site SQL Anywhere server to which the LTM connects. For Adaptive Server Enterprise during recovery, this is a data server with a temporary database whose logs the LTM will transfer to Replication Server. The LTM uses the SQL_log_files parameter to locate offline transaction logs.
SQL_user	The login name that the LTM uses to connect to the database specified by RS_source_ds and RS_source_db.

Example configuration file

The following is a sample LTM configuration file.

```
# This is a comment line
# Names are case sensitive.
SQL_user=SA
SQL_pw=sysadmin
SQL_server=PRIMESV
SQL_database=primedb
RS_source_ds=PRIMEOS
RS_source_db=primedb
RS=MY_REPSERVER
RS_user=sa
RS_pw=sysadmin
LTM_admin_user=DBA
LTM_admin_pw=sql
LTM_charset=cp850
scan_retry=2
SQL_log_files=e:\logs\backup
APC_user=sa
APC_pw=sysadmin
```

The Log Translation utility

With the Log Translation utility, you can translate a transaction log into a SQL command file.

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 incorporating into batch or command files.

Translate Log File wizard

The Translate Log File wizard helps you translate a log file into a *.sql* file from Sybase Central.

♦ To translate a transaction log into a command file (Sybase Central)

1. From the Tools menu, choose SQL Anywhere 10 ► Translate Log File.

   The Translate Log File wizard appears.

2. Follow the instructions in the wizard.

Log Translation utility (dbtran)

Syntax

Running against a database server:

dbtran [*options*]

Running against a transaction log:

dbtran [*options*] [*transaction-log*] [*SQL-file*]

Option	Description
@data	Read options from the specified environment variable or configuration file.
-a	Include uncommitted transactions.
-c *"keyword=value; …"*	Supply database connection parameters—cannot be used with a transaction log name.
-d	Display output in chronological order.
-ek *key*	Specify encryption key.
-ep	Prompt for encryption key.

Option	Description
-f	Output only since the last checkpoint.
-g	Include audit records in output.
-ir *offset1*,*offset2*	Include only the portion of the log between the two specified offsets.
-is *source*,…	Include only rows originating from the specified sources.
-it *user.table*,…	Include only operations on specified tables by specifying a comma-delimited list of *user.table*.
-j *date/time*	Show output from the last checkpoint prior to the given time.
-m	Specify transaction logs directory (requires -n option).
-n *filename*	Output SQL file, when used against a database server.
-o *filename*	Log output messages to a file.
-q	Run in quiet mode—do not display messages.
-r	Remove uncommitted transactions (default).
-rsu *username*,…	Override default Replication Server user names.
-s	Produce ANSI standard SQL UPDATE transactions.
-sr	Generate SQL Remote comments.
-t	Include trigger-generated transactions in output.
-u *userid*,…	Translate transactions for listed users only.
-x *userid*,…	Exclude transactions for listed users.
-y	Replace file without confirmation.
-z	Include trigger-generated transactions as comments only.
transaction-log	Log file to be translated.
SQL-file	Output file containing the translated information.

Description

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** Run in this way, 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 **demo10** and places the output in a file named *demo.sql*.

```
dbtran -c "ENG=demo10;DBN=demo;UID=DBA;PWD=sql" -n demo.sql
```

- **Against a transaction log file** Run in this way, 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, as well as 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.

☞ For more information about recovering with transactions that span multiple log files, see "Recovering from multiple transaction logs" on page 803.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

Log translation utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Include uncommitted transactions (-a)

The transaction log contains any 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 used, only committed transactions appear in the output file. If -a is used, any committed transactions found in the transaction log are output followed by a ROLLBACK statement.

Connection string (-c)

When running the utility against a database server, this parameter specifies the connection string.

DBA authority is required to run dbtran.

☞ For a description of the connection parameters, see "Connection parameters" on page 206.

Output in chronological order (-d)

Transactions are output in order from earliest to latest. This feature is provided primarily for use when auditing database activity: the output of this command should not be applied against a database.

Specify encryption key (-ek)

This option allows you to specify 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.

If you are running against a database server (using the -c option), make sure you specify the key using a connection parameter, not using the -ek option. For example, the following command gets the transaction log information about database *enc.db* from server **sample**, and saves its output in *log.sql*.

```
dbtran -n log.sql -c "ENG=sample;DBF=enc.db;UID=DBA;PWD=sql;DBKEY=mykey"
```

Prompt for encryption key (-ep)

This option allows you to specify in the command that you want to be prompted for the encryption key. This option causes a dialog box 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.

If you are running against a database server (using the -c option), make sure you specify the key using a connection parameter, not using the -ep option. For example, the following command gets the transaction log information about database *enc.db* from server **sample**, and saves its output in *log.sql*.

```
dbtran -n log.sql -c "ENG=sample;DBF=enc.db;UID=DBA;PWD=sql;DBKEY=mykey"
```

Output from last checkpoint only (-f)

Only transactions that were completed since the last checkpoint are output.

Include audit information (-g)

If the auditing database option is turned on, auditing information is added to the transaction log. You can include this information as comments in the output file using this option.

☞ For more information, see “auditing option [database]” on page 395.

The -g option implies the -a, -d, and -t options.

Include offset range (-ir)

Output a portion of the transaction log between two specified offsets.

Include rows from specified sources (-is)

Output 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**.

- **RepServer** Include only rows that were modified using the Replication Agent (LTM) and Replication Server. You can also use the short form **RS**.
- **Local** Include only rows that are not replicated.

Include specified tables (-it)

Output those operations on the specified, comma-separated list of tables. Each table should be specified as *owner.table*.

Output from the last checkpoint prior to a given date (-j)

Only transactions from the most recent checkpoint prior to the given date and/or time are translated. 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"*.

Transaction logs directory (-m)

Use this option to specify a directory that contains transaction logs. This option must be used in conjunction with the -n option.

Output file (-n)

When you run the dbtran utility against a database server, use this option to specify the output file that holds the SQL statements.

Log output messages to file (-o)

Write output messages to the named file.

Operate quietly (-q)

Do not display output messages. This option is available only when you run this utility from a command prompt. If you specify this option, you must also specify the -y option, or the operation will fail.

Do not include uncommitted transactions (-r)

Remove any transactions that were not committed. This is the default behavior.

Override Replication Server user names (-rsu)

By default, the -is option assumes the default Replication Server user names of dbmaint and sa. You can override this assumption using the -rsu option with a comma-separated list of user names.

Generate ANSI standard SQL UPDATE (-s)

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 in case of duplicate rows. If the option is used, the FIRST keyword is omitted for compatibility with the SQL standard.

Generate SQL Remote comments (-sr)

Place generated comments in the output file describing how SQL Remote distributes operations to remote sites.

Include transactions generated by triggers (-t)

By default, actions performed by triggers are not included in the command file. If the matching trigger is in place in the database, when the command file is run against the database, the trigger will perform 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 be run.

Output transactions for listed users only (-u)
This option allows you to limit the output from the transaction log to include only specified users.

Output transactions except for listed users (-x)
This option allows you to limit the output from the transaction log to exclude specified users.

Operate without confirming actions (-y)
Choosing this option automatically replaces existing command file(s) without prompting you for confirmation. If you specify -q, you must also specify -y or the operation will fail.

Include transactions generated by triggers as comments only (-z)
Transactions that were generated by triggers will be included only as comments in the output file.

transaction-log Log file to be translated. Cannot be used together with -c or -m options.

SQL-file Output file containing the translated information. For use with *transaction-log* only.

The Ping utility

The Ping utility is provided to assist in troubleshooting connection problems.

Ping utility (dbping)

Syntax

dbping [*options*]

Option	Description
@data	Read options from the specified environment variable or configuration file.
-c *"keyword=value; …"*	Specify database connection parameters.
-d	Make a database connection if the server is found.
-en	Exit with a failed return code if NULL is returned for any of the properties specified by -pc, -pd, or -ps.
-l *library*	Load specified ODBC driver or library.
-m	Use ODBC driver manager.
-o *filename*	Log output messages to a file.
-pc *property*,…	Report specified connection properties.
-pd *property*[*@db-name*],…	Report specified database properties.
-ps *property*,…	Report specified database server properties.
-q	Run in quiet mode—do not display messages.
-s	Report network statistics.
-st *time*	Report network statistics after running for the number of seconds specified.
-z	Display debugging information.

Description

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).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

Ping utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Connection parameters (-c)

Use this option to specify 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.

☞ For a description of the available connection parameters, see "Connection parameters" on page 206.

If you use the following command to start dbping and there is a database server named demo10 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 demo10 is found, dbping attempts to autostart one.

```
dbping -d -c "UID=DBA;PWD=sql;ENG=demo10;DBN=demo;DBF=samples-dir\demo.db"
```

☞ For information about the default location of *samples-dir*, see "The samples directory" on page 295.

Make database connection (-d)

Ping 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 "ENG=blair;DBN=demo"
```

The following command fails, with the message `Ping database failed -- specified database not found`:

```
dbping -c "ENG=blair;DBN=demo" -d
```

Exit with a failed return code if database properties return NULL (-en)

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. Specify the -en option if you want dbping to exit with a failed return code when NULL is returned for any of the properties specified. This option can only be used in conjunction with -pc, -pd, and -ps.

Load specified library (-l)

Specify the library to use (without its file extension). This option avoids the use of the ODBC driver manager, and so is particularly useful on Unix operating systems.

For example, the following command loads the ODBC driver directly:

```
dbping -m -c "DSN=SQL Anywhere 10 Demo" -l dbodbc10
```

On Unix, if you want to use a threaded connection library, you must use the threaded version of the Ping utility, dbping_r.

Use ODBC to connect (-m)

Establish a connection using ODBC. By default, the utility connects using the embedded SQL interface.

Log output messages to file (-o)

Write output messages to the named file.

Report connection properties (-pc)

Upon connection, display 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.

☞ For a list of connection properties, see “Connection-level properties” on page 476.

For example, the following command displays the divide_by_zero_error option setting, which is available as a connection property.

```
dbping -c ... -pc divide_by_zero_error
```

Report database properties (-pd)

Upon connection, display the specified database properties. Supply the properties in a comma-separated list.

☞ For a list of database properties, see “Database-level properties” on page 506.

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
```

Report database server properties (-ps)

Upon connection, display 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.

☞ For a list of database server properties, see “Server-level properties” on page 496.

For example, the following command displays the number of licensed seats or processors for the database server:

```
dbping -c ... -ps LicenseCount
```

Operate quietly (-q)

Do not display output messages.

Display network performance statistics (-s)

Use this option to obtain 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 desired 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 82.

Display network performance statistics for specified time (-st)

This option is the same as -s, except that it lets you specify 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 82.

Display debugging information (-z)

This option is available only when an embedded SQL connection is being attempted. That is, it cannot be combined with -m or -l. It displays the network communication protocols used to attempt connection, and other diagnostic messages.

The Rebuild utility

Databases can be rebuilt using the Rebuild batch file, command file, or shell script, which invokes a series of utilities to rebuild a database, or as part of the unload process using the Unload Database wizard in Sybase Central.

☞ For more information about the Unload Database wizard, see “The Unload utility” on page 689.

Rebuild utility (rebuild)

Syntax

rebuild *old-database new-database* [*DBA-password*]

Description

This batch file, command file, or shell script uses dbunload to rebuild *old-database* into *new-database*. This is a simple script, but it helps document the rebuilding process, and provides a basis for customization. Both database names should be specified without extensions. An extension of *.db* is automatically added.

You can use dbunload with the -ar option to perform unloading and reloading without using the rebuild batch file.

The *DBA-password* must be specified if the password for the DBA user in the *old-database* is not the initial password sql.

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.

Rebuild runs the dbunload, dbinit, and Interactive SQL commands with the default options. If you need different options, you must run the three commands separately.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, 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

- “Unload utility (dbunload)” on page 690
- “Initialization utility (dbinit)” on page 614
- “The Interactive SQL utility” on page 622

The Server Enumeration utility

The Server Enumeration utility is provided to assist in diagnosing connection problems by locating database servers on the immediate TCP/IP network.

Server Enumeration utility (dblocate)

Syntax

dblocate [*options*] [*server-name*]

Option	Description
@data	Read options from the specified environment variable or configuration file.
-d	Display the server name and address, and a comma-separated list of all databases running on each server.
-dn *database-name*	Display the server name and address only if the server is running a database with the specified name.
-dv	Display the server name and address, and list all databases running on each server on a separate line.
-n	Do not resolve IP addresses into computer names.
-o *filename*	Log output messages to a file.
-p *port-number*	Display servers using the specified TCP/IP port number.
-q	Run in quiet mode—do not display messages.
-s *name*	Display servers with the specified server name.
-ss *substr*	Display servers whose name contains *substr*.
-v	Display full server name for database servers with long names.
server-name	Specify a host name or IP address to list only servers running on the specified computer.

Description

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 database option” on page 200.

Depending on your network, it may take several seconds for dblocate to prints its results.

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 protocol option [PORT]" on page 258.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, 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.

☞ For more information about LDAP, see "Connecting using an LDAP server" on page 108.

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 in cases where 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 Enumeration utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

List all running databases (-d)

For each server found, list the server name and address, 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 database option" on page 197.

Display servers running a database with the specified name (-dn)

List 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 database option" on page 197.

List all running databases on a separate line (-dv)

For each server found, display the server name and address, 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 database option" on page 197.

Do not resolve IP addresses into computer names (-n)
List IP addresses in the output, rather than computer names. This may improve performance since looking up computer names may be slow.

Log output messages to file (-o)
Write output messages to the named file.

Display servers running on the specified port (-p)
Display 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.

Operate quietly (-q)
Do not display output messages.

Display servers with the specified name (-s)
Display 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).

Display servers whose name includes the specified substring (-ss)
Display 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).

Display full server name for servers with long names (-v)
By default, dblocate truncates database server names that are longer than 40 bytes. Specify -v if you want dblocate to display the full server name.

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.

server-name
List 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.

The Server Licensing utility

The Server Licensing utility licenses seats or processors for your network database server.

Server Licensing utility (dblic)

Syntax

dblic [*options*] *executable-name* **"***user-name***"** **"***company-name***"**

Option	Description
@data	Read options from the specified environment variable or configuration file.
-l *type*	Specify the license type. Allowed values are **perseat** or **processor**.
-o *filename*	Log output messages to a file.
-q	Run in quiet mode—do not display messages.
-u *license-number*	Specify the total number of users or processors for license.
executable-name	Specify the database server executable to be licensed, with path relative to the current directory.
user-name	Specify the user name for the license.
company-name	Specify the company name for the license.

Description

The Server Licensing utility adds licensed users or licensed processors to your network database 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 you license.

This utility also modifies the user and company names displayed at startup by the personal or network database servers.

You can also use this utility to view the current license information for a personal or network database server by entering only the executable name.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, 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 will fail. Make sure the executable is writable (for example, using `chmod +w`) before you use the Server Licensing utility.

The Server Licensing utility is not supported on NetWare. To license SQL Anywhere executables on NetWare, you must make sure that the SQL Anywhere database server on NetWare is not running, and then run the Server Licensing utility from a Windows computer that can access the NetWare volume containing the SQL Anywhere software.

☞ For more information about SQL Anywhere licensing, visit http://www.ianywhere.com/products/sa_licensing.html.

Server Licensing utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

License type (-l)

Enter 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. You can use any or all of the processors on the computer running the database server.
- **Processor** A processor license restricts the number of separate physical processors that can be used by the database server. The database server treats each physical processor as a CPU for the purposes of this license type, 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.

Log output messages to file (-o)

Write output messages to the named file.

Operate quietly (-q)

Do not display output messages.

License number (-u)

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.

executable-name

Enter the path and file name of the personal database server executable (**dbeng10.exe**) or network database server executable (**dbsrv10.exe**) you are licensing.

You can view the current license information for a server executable by entering only the executable name.

user-name

A user name for the license. This name appears on the Server Messages window on startup. If there are spaces in the name, enclose it in double quotes.

company-name

The company name for the license. This name appears on the Server Messages window on startup. If there are spaces in the name, enclose it in double quotes.

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 Windows XP network database server. The command must be entered all on one line:

```
dblic -l perseat -u 50 dbsrv10.exe "Sys Admin" "My Co"
```

The following messages appear on the screen to indicate the success of the license:

```
Licensed nodes: 50
User: Sys Admin
Company: My Co
```

The following command returns information about the license for a database server:

```
dblic dbsrv10.exe
```

The Service utility

The Service utility is a tool used to create, delete, and modify SQL Anywhere services. You can access the Service utility in the following ways:

- From Sybase Central, using the Create Service wizard.
- At a command prompt, using the dbsvc command.

Create Service wizard

The Create Service wizard helps you create a new Windows service from Sybase Central.

♦ **To create a service (Sybase Central)**

1. From the Context dropdown list, choose the SQL Anywhere 10 plug-in.
2. In the right pane, click the Services tab.
3. From the File menu, choose New ▶ Service.

   The Create Service wizard appears.

Tip
You can also create services for the MobiLink plug-in. See "Adding, modifying, and removing services" [*MobiLink - Server Administration*].

Service utility (dbsvc) for Windows

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*]

Major option	Description
@data	Read options from the specified environment variable or configuration file.
-d *service-name*	Delete a service.
-g *service-name*	Get details of a service.
-l	List all SQL Anywhere services.
-u *service-name*	Start the service named *service-name*.
-w *executable parameters*	Create a service.
-x *service-name*	Stop the service named *service-name*.

Creation option	Description
-a *acct*	Specify the account name to use (used with -p).
-as	Use local system account.
-i	Allow service to interact with desktop.
-p	Specify a password for the account (used with -a).
-rg *dependency,…*	Specify group dependencies when creating a service.
-rs *dependency,…*	Specify service dependencies when creating a service.
-s *startup*	Specify the startup option (the default is manual)—you must specify one of automatic, manual, or disabled.
-sd *description*	Provide a description of the service.
-sn *name*	Specify a name for the service.
-t *type*	Specify the type of service.

Modifier option	Description
-cm	Display the service creation command (must use with -g or -l).
-o *log-file*	Write output from the dbsvc utility to the specified log file.
-q	Do not display messages.
-y	Delete or overwrite service without confirmation.

Description

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. The Service utility provides the same functionality as the Create Service wizard in Sybase Central.

You must be a member of the Administrators group on the local computer to use the Service utility.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

☞ For more information about services, see "Understanding Windows services" on page 34.

Service utility options

When you use the Service utility, you can choose from major options, modifier options, and details options.

Major options

Specify environment variable or configuration file (@*data*)
Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Delete a service (-d)
Removes the named service from the list of services. If you supply -y, the service is deleted without confirmation.

Get details of a service (-g)
Lists the definition of the service, not including the password.

List all SQL Anywhere services (-l)
Lists the available SQL Anywhere services.

Start a service (-u)
Starts a service named *service-name*.

Create service (-w)
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 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 directory in its path.

You must supply parameters appropriate for the service you are creating.

For more information, see:

- ♦ **dbsrv10 and dbeng10** "The SQL Anywhere database server" on page 118.
- ♦ **mlsrv10** "MobiLink Server Options" [*MobiLink - Server Administration*].

- **dbmlsync** “dbmlsync syntax” [*MobiLink - Client Administration*].
- **dbremote** “Message Agent” [*SQL Remote*].
- **dbns** “SQL Anywhere Broadcast Repeater utility (dbns10)” on page 665
- **dbltm** “Log Transfer Manager utility (dbltm)” on page 631

Stop a service (-x)

Stops a service named *service-name*.

Creation options

Account name (-a)

All services run under a 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.

Use local system account (-as)

All services run under a Windows account. When -as is specified, the service runs under the Windows LocalSystem account. No password is required. One of -a or -as must be used.

Allow service to interact with desktop (-i)

Displays an icon that you can double-click to display the Server Messages window.

Specify password for account (-p)

Use this option with the -a option to specify the password for the account under which the service runs.

Set group dependencies (-rg)

Specifies one or more load ordering groups that must be started before the service being created is allowed to start.

Set service dependencies (-rs)

All the services in the list must have started before the service being created is allowed to start.

Set startup option (-s)

Sets startup behavior for SQL Anywhere services. You can set startup behavior to Automatic, Manual, or Disabled. The default is Manual.

Provide service description (-sd)

Use this option to provide a description of the service. The description appears in the Windows Service Manager.

Provide service name (-sn)

Use this option to provide 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 10\win32\dbeng10.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 10\win32\dbeng10.exe"
```

Specify type of service (-t)

Specifies the type for this service. You can choose from the following types:

Type	Description
Network	SQL Anywhere network database server (dbsrv10). See "The SQL Anywhere database server" on page 118.
Standalone	SQL Anywhere personal database server (dbeng10). See "The SQL Anywhere database server" on page 118.
DBLTM	SQL Anywhere Log Transfer Manager (LTM). You can also specify **LTM** for this service type. See "Log Transfer Manager utility (dbltm)" on page 631.
DBRemote	SQL Remote Message Agent (dbremote). See "Message Agent" [*SQL Remote*].
MobiLink	MobiLink server (mlsrv10). See "mlsrv10 syntax" [*MobiLink - Server Administration*].
dbmlsync	MobiLink synchronization client (dbmlsync). See "dbmlsync syntax" [*MobiLink - Client Administration*].
DBNS	SQL Anywhere Broadcast Repeater. See "SQL Anywhere Broadcast Repeater utility (dbns10)" on page 665.
dblsn	Listener utility. See "Listener Utility" [*MobiLink - Server-Initiated Synchronization*].

The default setting for all service types is Standalone.

Modifier options

Display the service creation command (-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.

Log output to a file (-o)

Writes output from the Service utility (dbsvc) to the specified file. The -o option *must* occur 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\win32\dbsrv10 -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*.

Do not display messages (-q)

Do not display messages in the 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.

Delete or overwrite service without confirmation (-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.

Examples

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 10\win32\dbeng10.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
10\win32\dbsrv10.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 Workstation -rg TDI -w ...
```

Create a service called mysyncservice:

```
dbsvc -as -s manual -t dbmlsync -w mysyncservice "c:\Program Files\SQL
Anywhere 10\win32\dbmlsync.exe" -c "SQL Anywhere 10 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 10\win32\dbeng10.exe"
```

Create a MobiLink listener service that is started manually:

```
dbsvc -as -i -w myListener "c:\Program Files\SQL Anywhere 10\win32\dblsn.exe"
"@c:\temp\dblsn.opt"
```

Start the myListener service:

```
dbsvc -u myListener
```

Stop the myListener service:

```
dbsvc -x myListener
```

Service utility (dbsvc) for Linux

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*]

Major option	Description
-d *service-name*	Delete a service.
-g *service-name*	Get details of a service.
-l	List all SQL Anywhere services.
-u *service-name*	Start the service named *service-name*.
-w *executable parameters*	Create a service.
-x *service-name*	Stop the service named *service-name*.

Creation option	Description
-a *acct*	Specify account name to use.
-as	Use daemon account.
-t *type*	Specify the type of service.

Modifier option	Description
-cm	Display the service creation command (must use with -g or -l).
-q	Do not display messages.
-y	Delete or overwrite service without confirmation.

Linux-specific options	Description
-od	Specify the location of the system information file (if required).
-pr	Set the nice level for the Linux process.
-rl	Specify the runlevels on which to start the service.
-rs	Specify service dependencies when creating a service.
-status	Return the state the service is running.

Description

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 recommended that you do not use spaces in data source names.

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 SA_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
```

Service utility options

When you use the Service utility, you can choose from major options, modifier options, and details options.

Major options

Delete a service (-d)

Removes the named service from the list of services. If you supply -y, the service is deleted without confirmation.

Get details of a service (-g)

Lists the definition of the service.

List all SQL Anywhere services (-l)

Lists the available SQL Anywhere services.

Start a service (-u)

Starts a service named *service-name*.

Create service (-w)

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 must supply parameters appropriate for the service you are creating.

For more information, see:

- **dbsrv10 and dbeng10** "The SQL Anywhere database server" on page 118.
- **mlsrv10** "MobiLink Server Options" [*MobiLink - Server Administration*].
- **dbmlsync** "dbmlsync syntax" [*MobiLink - Client Administration*].
- **dbremote** "Message Agent" [*SQL Remote*].

Stop a service (-x)

Stops a service named *service-name*.

Creation options

Account name (-a)

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.

The Login as a Service privilege is required for all accounts other than the daemon account.

Use daemon account (-as)

All services run under a Linux account. When -as is specified, the service runs under the Linux daemon account. No password is required. One of -a or -as must be used.

Set service dependencies (-rs)

All the services in the list must have started before the service being created is allowed to start. Linux services are dependent on base services. For example, if service A relies on service B, and service B relies on service C, SQL Anywhere does not attempt to start service C.

Specify type of service (-t)

Specifies the type for this service. You can choose from the following types:

Type	Description
Network	SQL Anywhere network database server (dbsrv10). See "The SQL Anywhere database server" on page 118.
Standalone	SQL Anywhere personal database server (dbeng10). See "The SQL Anywhere database server" on page 118.
DBRemote	SQL Remote Message Agent (dbremote). See "Message Agent" [*SQL Remote*].
MobiLink	MobiLink server (mlsrv10). See "mlsrv10 syntax" [*MobiLink - Server Administration*].
dbmlsync	MobiLink synchronization client (dbmlsync). See "dbmlsync syntax" [*MobiLink - Client Administration*].

The default setting for all service types is Standalone.

Modifier options

Display the service creation command (-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.

Do not display messages (-q)

Suppress messages to the console. If you specify this option when modifying or deleting an existing service, you must also specify -y or the operation will fail.

Delete or overwrite service without confirmation (-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.

Examples

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 -o syncinfo.txt -w mysyncservice -c "/tmp/CustDB.db"
```

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
```

The SQL Anywhere Broadcast Repeater utility

The SQL Anywhere 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.

♦ To use the SQL Anywhere Broadcast Repeater

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 servers on the remote subnet, and clients are able to connect to 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.

SQL Anywhere Broadcast Repeater utility (dbns10)

Syntax

dbns10 [*options*] [*address* ...]

Options

Option	Description
@data	Read options from the specified environment variable or configuration file.
-ap *port*	Specify the SQL Anywhere port number to use (2638 used by default).
-m *ip*	Specify the IP address or computer name to listen on. This option is required for computers with more than one IP address.
-o *file*	Write output to the specified file.

Option	Description
-p *port*	Specify the port number to use (3968 used by default).
-q	Run in quiet mode—do not display messages.
-s	Ensure no other dbns processes are running on this subnet.
-x	Shut down dbns processes at specified addresses.
-z	Run in debug mode.
address	Specify the IP address or host name of other computers that are, or will be, running DBNS processes.

Description

The SQL Anywhere 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 all 32-bit and 64-bit Windows platforms, as well as all supported Unix platforms.

The clients and database server must be running SQL Anywhere 9.0.2 or later to use the SQL Anywhere Broadcast Repeater.

Caution
It is recommended that you do not run the dbns10 utility on the same computer as a SQL Anywhere database server because it is possible that dbns10 or the database server may not receive UDP broadcasts.

SQL Anywhere Broadcast Repeater options

Specify addresses of other DBNS processes (*address*) Use this option to specify the IP address or host name of other computers that are, or will be, running DBNS processes. This allows the DNBS processes to detect each other and exchange information about known database servers and other DBNS processes.

Specify environment variable or configuration file (@*data*)
Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Specify the port number to use (-ap)
Specify the port number used by the database server. The default port number is 2638.

Specify the IP address to listen on (-m)
Specify 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.

Log output messages to file (-o)

Write the output that appears in the SQL Anywhere Broadcast Repeater Messages window to the named file.

Specify the port number to use (-p)
Use this option to specify 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.

Operate quietly (-q)
Do not display output messages.

Ensure no other DBNS processes are running on the same subnet (-s)
When you start the SQL Anywhere Broadcast Repeater with this option specified, the new DBNS process checks if another DBNS process is already running on that subnet, and returns an error before shutting down if another DBNS process is found.

Shut down dbns process at the specified address (-x)
Specify this option to shut down the DBNS process running on specified host. You can specify an IP address or host name.

Run in debug mode (-z)
When running in debug mode, a line appears in the SQL Anywhere 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.

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 machine on the 10.50.125.255 subnet (Computer B at 10.50.125.103).

On each of these two computers, run dbns10, passing the IP address of the other computer. Execute the following command on Computer A:

```
dbns10 10.50.125.103
```

On Computer B, execute the following command:

```
dbns10 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:

```
dbns10 -m 10.50.83.114  10.50.125.103
```

The SQL Anywhere Console utility

The SQL Anywhere Console provides administration and monitoring facilities for database server connections.

The SQL Anywhere Console is available on all supported platforms except Windows CE, AIX, HP-UX, HP-UX Itanium, and Linux 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 XP/200x, Mac OS X, or Linux).

☞ For more information about obtaining property values, see "Database Properties" on page 475.

SQL Anywhere Console utility (dbconsole)

Syntax

dbconsole [*options*]

Option	Description
@data	Read options from the specified environment variable or configuration file.
-c *"keyword=value; ..."*	Specify database connection parameters.
-datasource *DSN-name*	Specify an ODBC data source to connect to.
-host *hostname*	Specify the *hostname* or IP address of the computer running a database server.
-port *port-number*	Look on the specified port number for the database server.

Description

The SQL Anywhere Console allows you to monitor the server from a client computer. You can use it to track who is logged on to a database server elsewhere on your network. You can also display both 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.

♦ To disconnect a user from a database

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

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 dialog, which can be accessed by choosing File ▶ Options.

Console utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

-c "keyword=value; ..."

Specify connection parameters.

☞ For a description of supported connection parameters, see "Connection parameters" on page 206.

Data source (-datasource)

Specify an ODBC data source to connect to. You do not need to be using the iAnywhere JDBC driver to use this option.

Host (-host)

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.

Database server port (-port)

Specify the port number on which the database server is running. The default port number for SQL Anywhere is **2638**.

The SQL Anywhere Script Execution utility

The SQL Anywhere Script Execution utility (dbrunsql) allows you to type SQL commands or run command files on Windows CE.

SQL Anywhere Script Execution utility (dbrunsql)

Syntax

dbrunsql [*options*] [*SQL-script-file* | *SQL-command*]

Option	Description
-c *"keyword=value; ..."*	Specify database connection parameters.
-d	Allow only data exported from result sets to be written to the output file.
-e [**c** | **p** | **s**]	Specify the behavior when an error occurs: ♦ **c** Continue. ♦ **p** Prompt. This is the default setting. ♦ **s** Stop.
-f [**f** | **a**]	Choose FIXED (the default) or ASCII data format when exporting result sets.
-g [**+** | **-**]	Specify whether the GUI is enabled on Windows CE (it is enabled by default).
-o *filename*	Log output messages to a file.
-q	Run in quiet mode—do not display messages or windows. This does not suppress error messages.
-qc	Close window on completion.
-s *number*	Specify the maximum number of bytes fetched per column when exporting FIXED format (the default value is 255).
-v	Run in verbose mode: display all lines of each SQL statement.

Description

The dbrunsql utility allows you to type SQL commands or run command files against a database. The SQL Anywhere Script Execution utility (dbrunsql) is only supported on Windows CE.

Options

Connection parameters (-c)

Specify connection parameters. See "Connection parameters" on page 206 for a description of the connection parameters.

Write result set data to file (-d)
When this option is specified, only data exported from result sets is written to the output file. If you do not specify -d, then all dbrunsql output is written to the output file.

Specify behavior when an error occurs (-e) This option controls what happens if 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.

Specify data format for exporting result sets (-f)
Use this option to specify 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.

Control whether dbrunsql GUI appears (-g)
By default, the dbrunsql GUI appears. Specify -g- if you do not want the GUI to appear.

Log output messages to file (-o)
Write output messages to the named file.

Operate quietly (-q)
Do not display 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

Close window on completion (-qc)
Specify this option if you want the dbrunsql window to close once the command or script file has been executed.

Specify the maximum number of bytes fetched per column (-s)
When you are exporting result sets using the FIXED format, you can specify the maximum number of bytes fetched per column. The default value is 255.

Run in verbose mode (-v)
When you specify the -v option, all lines of each SQL statement appear in the dbrunsql output. Otherwise, when you execute a script file, the number of the line that is currently being executed appears.

The SQL Anywhere Support utility

The SQL Anywhere Support utility sends information about errors and software usage to iAnywhere.

You can configure the SQL Anywhere Support utility so that 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 server (mlsrv)
- network server (dbsrv10)
- personal server (dbeng10)
- QAnywhere agent (qaagent)
- Replication Agent (dbltm)
- SQL Anywhere client for MobiLink (dbmlsync)
- SQL Anywhere Console utility (dbconsole)
- SQL Remote (dbremote)
- Sybase Central

You can also use the SQL Anywhere Support utility (dbsupport) to send feature statistics and unsubmitted crash reports at any time, or to view the status of submitted information.

SQL Anywhere Support utility (dbsupport)

Syntax

dbsupport [*options*] *operation* [*operation-specific-option*]

Option	Description
@data	Read options from the specified environment variable or configuration file.
-o *filename*	Send output to the specified file.
-q	Do not display messages.

Operations	Description
-is *submission-ID* [**-rr** *N*]	Inquire about the status of the specified submission.
-iu [**-r** *N*]	Inquire about software updates.
-lc	List crash reports that have not yet been submitted.
-ls	List submission IDs for crash reports that have been submitted.
-pc *filename*	Print crash report file information.

Operations	**Description**
-pd	Print current diagnostic information that has been gathered.
-ps *submission-ID*	Print submission information for the specified submission.
-sa [**-r** *number-of-submission-retries*]	Submit all reports and diagnostic information.
-sc *reportname* [**-r** *number-of-submission-retries*] [**-nr** | **-rr** *N*]	Submit crash report file and associated files.
-sd [**-r** *number-of-submission-retries*]	Submit diagnostic information.

Configuration option	**Description**
-cc [**autosubmit** | **no** | **promptDefY** | **promptDefN**]	Configure auto-submission of crash reports.
-cd	Configure the default submission retry delay.
-ch *crash-handler-program*	Configure crash handler program.
-ch-	Remove the crash handler setting.
-cid *customer-id*	Configure a customer identification string.
-cid-	Remove the customer identification string.
-cp *email-server* [**:***port*]	Configure HTTP proxy host and port.
-cp autodetect	Attempt to autodetect and configure the HTTP proxy host and port (Windows only).
-cp-	Remove the HTTP port and proxy host settings.
-cr *number-of-submission-retries*	Configure default submission retry count (10 by default).

Operation-specific option	**Description**
-nr	Do not check for a response to a submission.
-r *number-of-submission-retries*	Retry the operation the specified number of times. (0 means retry indefinitely).
-rd *retry-delay*	Retry checking for a submission response, waiting the specified number of seconds between attempts.
-rr *number-of-submission-response-retries*	Retry the specified number of times for a submission response.

Description

The SQL Anywhere Support utility (dbsupport) can be used for any of the following tasks:

- submit diagnostic information to iAnywhere over the Internet
- list information about submitted and unsubmitted reports
- print information about submitted and unsubmitted 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 one a database or synchronization server

The SQL Anywhere Support utility allows you to submit the following types of diagnostic information to iAnywhere over the Internet:

- database and MobiLink server error reports
- database and MobiLink server usage statistics
- administration tool (Sybase Central, Interactive SQL, SQL Anywhere console utility) error reports

When a report is successfully submitted, it is assigned a unique submission ID. Reports are written to the diagnostic directory.

☞ For information about the diagnostic directory, see "SADIAGDIR environment variable" on page 283.

☞ For more information about error reports and how they are submitted, see "Error reporting in SQL Anywhere" on page 46.

Platform	Default diagnostic directory location
Windows (except Windows CE)	*%ALLUSERSPROFILE%\Application Data\SQL Anywhere 10 \diagnostics*
Windows CE	Directory where the executable is running.
Unix	*$HOME/.sqlanywhere10/diagnostics*
NetWare	*SYS:\SYSTEM*

For more information about the diagnostic directory, see "SADIAGDIR environment variable" on page 283.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

The SQL Anywhere Support utility can be configured to perform certain actions when a problem is detected. For example, it could 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.

The SQL Anywhere 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 SQL Anywhere Support utility are stored in the *dbsupport.ini* file in the diagnostic directory.

SQL Anywhere Support utility options

Specify environment variable or configuration file (@*data*)
Use this option to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Log output messages to file (-o)
Write the output to the named file.

Operate quietly (-q)
Display only critical messages.

Configuration options

Configure automatic report submission (-cc)
The -cc option lets you change 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 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 SQL Anywhere Support utility to not submit reports to iAnywhere.

If you specify this option, its value becomes the default used by the SQL Anywhere Support utility. The configuration is stored in the *dbsupport.ini* file in the diagnostic directory.

The following command configures the SQL Anywhere Support utility so that it does not submit reports and never prompts the user to submit reports:

```
dbsupport -cc no
```

Specify submission retry delay (-cd)

Specify -cd to specify 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 SQL Anywhere Support utility. The configuration 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
```

Configure crash handler program (-ch)

The -ch option is used to specify a program that is called when a crash occurs.

If you specify this option, its value becomes the default used by the SQL Anywhere Support utility. The configuration 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 substituted for the full path to the location of the generated report file.
- **%P** This parameter is substituted for the name of the program that generated the report. For example, if a version 10 personal database server generates the report, dbeng10 is returned.
- **%S** This parameter is substituted for 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.

> **Note**
> 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 would be 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 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, as shown above.

Remove crash handler settings (-ch-)

Use the -ch- option to remove the crash handler settings that are stored in the *dbsupport.ini* file. For example:

```
dbsupport -ch-
```

Configure a customer identification string (-cid)

The -cid option specifies a string that identifies you in the submission report. If you specify this option, its value becomes the default used by the SQL Anywhere 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"
```

Remove customer identification string (-cid-)

Use the -cid- option to remove the customer identification string from the *dbsupport.ini* file. For example:

```
dbsupport -cid-
```

Configure HTTP proxy host and port (-cp)

Use the -cp option to configure the HTTP proxy host and port used to submit crash 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

Remove HTTP proxy host and port settings (-cp-)

Use the -cp- option to remove HTTP proxy host and port settings from the *dbsupport.ini* file. For example:

```
dbsupport -cp-
```

Specify maximum number of submission retries (-cr)

Use the -cr option to specify the number of times a failed submission should be retried.

If you specify this option, its value becomes the default used by the SQL Anywhere Support utility. The configuration is stored in the *dbsupport.ini* file in the diagnostic directory.

The following dbsupport command specifies that failed operations are should be retried every 3 seconds, up to a maximum of 4 times before giving up:

```
dbsupport -cr 4 -cd 3
```

Operations options

Inquire about submission status (-is)

Use the -is option to check the status of a crash report that has been submitted to iAnywhere.

For example, the following command inquires about the status of submission ID 66:

```
dbsupport -is 66
```

Inquire about software updates (-iu)

You can specify the -iu option to check for updates to your build of SQL Anywhere.

You can also check for updates using Interactive SQL and Sybase Central. See "Options dialog: Check for Updates tab" [*SQL Anywhere 10 - Context-Sensitive Help*].

List unsubmitted crash reports (-lc)

Use the -lc option to generate a list of all crash reports that have not been submitted to iAnywhere. The report names listed can be used with the -sc option.

List submission IDs (-ls)

Specify the -ls option to generate a list of submission IDs for all reports that have been submitted to iAnywhere. For example:

```
dbsupport -ls
```

This returns information similar to the following:

```
Submission ID: 4
Minicore dump 20051220_133828_32116 reported: Wed Mar 15 16:31:56 2006
Submission ID: 98
Minicore dump 20051229_221211_3221 reported: Wed Mar 22 16:33:26 2006
```

Print crash report IDs (-pc)

Specify the -pc option to print crash report information. You can use this option to view information before it is submitted to iAnywhere.

Print diagnostic information (-pd)

Specify the -pd option to print the diagnostic information that has been collected. You can use this option to view information before it is submitted to iAnywhere.

Display information about submitted reports (-ps)

Use the -ps option to display information about a specific report that has been submitted to iAnywhere. For example:

```
dbsupport -ps 4
```

This returns information about submission 4:

```
Minicore dump 20051220_133828_32116 reported: Wed Mar 15 16:31:56 2006
```

Submit all crash reports and diagnostic information (-sa)

Specify the -sa option to submit all crash report and diagnostic information stored in the diagnostic directory to iAnywhere.

Submit crash report and related files (-sc)

Specify the -sc option to submit a crash report and diagnostic information to iAnywhere. For example:

```
dbsupport -sc 20051220_133828_32116
```

Use the -lc option to see a list of reports that have not been submitted.

Submit diagnostic information (-sd)

Specify the -sd option to submit all crash report and diagnostic information to iAnywhere.

☞ For information about the diagnostic directory, see "SADIAGDIR environment variable" on page 283.

Operation-specific options

The SQL Anywhere Support utility supports the following operation-specific options. These are useful for overriding default behavior, including those that have been saved in the *dbsupport.ini file*:

Do not check server for a response (-nr)

If you specify -nr, then 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 20051220_133828_32116
```

By default, dbsupport checks whether there is already a fix for the problem being submitted.

Specify the number of submission attempts (-r)

Use the -r option to specify 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* if one exists.

Specify delay between submission attempts (-rd)

Use the -rd option to specify 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* if one exists.

Retry submission (-rr)

Use the -rr option to specify the maximum number of times dbsupport should try to obtain a submission response. Specifying 0 means retry indefinitely. The default value is 10.

The Start Server in Background utility

This utility is provided to start a database server in the background.

Start Server in Background utility (dbspawn)

Syntax

dbspawn [*options*] *server-command*

Option	Description
@data	Read options from the specified environment variable or configuration file.
-f	Do not check for a running server.
-p	Report operating system process ID.
-q	Run in quiet mode—do not display messages.
server-command	Specify the command line for starting the database server.

Description

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*].

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 dbeng10 "c:\my databases\mysalesdata.db"
```

If the specified path does not contain spaces, then quotes are not required.

Start Server in Background utility options

Specify environment variable or configuration file (@*data*)
Use this option to read 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) does not expand the contents of configuration files specified with the @data option.

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Do not check for a running server (-f)
The dbspawn utility will not start a new database server if it detects that a database server is already running on your system. This option prevents dbspawn from checking for existing database servers before attempting to start the database 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.

Report process ID (-p)
The operating system process ID of the database server process. For example:

```
dbspawn -p dbeng10 -n newserver
```

reports a message of the following form to a command prompt:

```
New process ID is 306
```

Operate quietly (-q)
Do not display output messages.

server-command Specifies the command line for starting the database server.

☞ For a description of the server commands, see "The SQL Anywhere database server" on page 118.

The Stop Server utility

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 Shutdown on the Server Messages window.

The Stop Server utility (dbstop) is not available on NetWare.

Stop Server utility (dbstop)

Syntax

dbstop [*options*] [*server-name*]

Option	Description
@data	Read options from the specified environment variable or configuration file.
-c *"keyword=value*; …*"*	Specify connection parameters.
-d	Stop specified database only.
-o *filename*	Log output messages to a file.
-q	Run in quiet mode—do not display messages.
-x	Do not stop if there are active connections.
-y	Stop without prompting even if there are active connections.
server-name	Specify the name of a local database server to stop.

Description

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 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).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

The Stop [dbstop] utility is not available on NetWare.

If you want to use the SQLCONNECT environment variable with dbstop, you should specify the -c option. Otherwise, you can get unexpected results.

Stop Server utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Connection parameters (-c)

When stopping a network server, you must supply a connection string with a user ID that has permissions to stop the server. By default, DBA permission 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.

☞ For more information, see "Connection parameters" on page 206, "Unconditional connection parameter [UNC]" on page 239, and "-gk server option" on page 148.

Stop database only (-d)

Do not stop the database server. Instead, only stop the database specified in the connection string.

Log output messages to file (-o)

Write output messages to the named file.

Operate quietly (-q)

Do not display a message if the database was not running.

Do not stop if there are active connections (-x)

Do 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.

Stop without prompting (-y)

Stop the server even if there are still active connections to the server. This is equivalent to including Unconditional=YES in te connection parameters.

server-name

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.

If you supply a server name, do not supply connection parameters as well.

☞ For more information, see "-gk server option" on page 148.

Examples

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;ENG=myserver;DBN=utility_db"
```

You are running the server named myserver with the database *demo.db* started. To stop the server and database:

```
dbstop -c "UID=DBA;PWD=sql;ENG=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;ENG=myserver;DBN=demo" -d
```

The Transaction Log utility

With the Transaction Log utility, you can 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).

You can access the Transaction Log utility in the following ways:

- From Sybase Central, using the Change Log File Settings wizard.
- From Interactive SQL, using the ALTER DATABASE *dbfile* ALTER LOG statement.

  For more information, see “ALTER DATABASE statement” [*SQL Anywhere Server - SQL Reference*].
- At a command prompt, using the dblog command.

Change Log File Settings wizard

The Change Log File Settings wizard helps you change the settings for a database log file from Sybase Central.

♦ **To change a transaction log file name (Sybase Central)**

1. From the Tools menu, choose SQL Anywhere 10 ► Change Log File Settings.

   The Change Log File Settings wizard appears.
2. Follow the instructions in the wizard.

Transaction Log utility (dblog)

Syntax

dblog [*options*] *database-file*

Option	Description
@data	Read options from the specified environment variable or configuration file.
-ek *key*	Specify encryption key.

Option	Description
-ep	Prompt for encryption key.
-g *n*	Sets the LTM generation number to *n*.
-il	Ignore the LTM truncation offset stored in the database.
-ir	Ignore the SQL Remote truncation offset stored in the database.
-is	Ignore the dbmlsync truncation offset stored in the database.
-m *mirror-name*	Set transaction log mirror name.
-n	No longer use a transaction log or mirror log.
-o *file-name*	Log output messages to a file.
-q	Run in quiet mode—do not display messages.
-r	No longer use a transaction log mirror.
-t *log-name*	Set the transaction log name.
-x *n*	Set the transaction log current relative offset to *n*.
-z *n*	Set the transaction log starting offset to *n*.

Description

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.

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 mirror log file, if any
- The current relative offset

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see “Software component exit codes” [*SQL Anywhere Server - Programming*].

Transaction log utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see “Using configuration files” on page 587.

Specify encryption key (-ek)

This option allows you to specify 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.

Prompt for encryption key (-ep)

This option allows you to specify that you want to be prompted for the encryption key. This option causes a dialog box 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.

Set the generation number (-g)

Use this option if you are using the Log Transfer Manager to participate in a Replication Server installation. It can be used after a backup is restored, to set the generation number. It performs the same function as the following Replication Server function:

```
dbcc settrunc( 'ltm', 'gen_id', n )
```

☞ For information about generation numbers and dbcc, see your Replication Server documentation.

Ignore the LTM truncation offset (-il)

This option can be used if you have stopped using the Log Transfer Manager to participate in a Replication Server installation on this database, but continue to use SQL Remote or MobiLink synchronization. It resets the Log Transfer Manager log offset that is kept for the delete_old_logs option, allowing transaction logs to be deleted when they are no longer needed.

It performs the same function as the following Replication Server function:

```
dbcc settrunc( 'ltm', 'ignore' )
```

☞ For information about dbcc, see your Replication Server documentation.

Ignore the SQL Remote truncation offset (-ir)

This option can be used if you have stopped using SQL Remote on this database, but continue to use the Log Transfer Manager or MobiLink synchronization. It 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.

Ignore the dbmlsync truncation offset (-is)

This option can be used if you have stopped using MobiLink synchronization on this database, but continue to use the Log Transfer Manager or SQL Remote. It 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.

Set the name of the transaction log mirror file (-m)

This option sets a 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.

No longer use a transaction log (-n)

Stop using a transaction log, and stop using a mirror log. 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, Log

Transfer Manager, or dbmlsync truncation offset exists, the transaction log cannot be removed unless the corresponding ignore option (-il for the Log Transfer Manager, -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).

Log output messages to file (-o)

Write output messages to the named file.

Operate quietly (-q)

Do not display output messages.

No longer use a transaction log mirror (-r)

For databases that maintain a mirrored transaction log, this option changes their behavior to maintain only a single transaction log.

Set the name of the transaction log file (-t)

This option sets a 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.

Set the current log offset (-x)

For use when reloading a SQL Remote consolidated database. This option resets the current log offset so that the database can take part in replication.

☞ For information on how to use this option, see "Unloading and reloading a database participating in replication" [*SQL Remote*].

Set the starting log offset (-z)

For use when reloading a SQL Remote consolidated database. This option resets the starting log offset so that the database can take part in replication.

☞ For information on how to use this option, see "Unloading and reloading a database participating in replication" [*SQL Remote*].

The Unload utility

Upgrading to version 10
For detailed information about rebuilding an existing database into a version 10 database, see “Upgrading SQL Anywhere” [*SQL Anywhere 10 - Changes and Upgrading*].

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 of 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.

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 dialog in Sybase Central.

☞ For more information, see “Using the Unload Data dialog” [*SQL Anywhere Server - SQL Usage*].

Accessing the Unload utility

You can access the Unload utility in the following ways:

- From Sybase Central, using the Unload Database wizard.
- At a command prompt, using the dbunload command. This is useful for incorporating 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 the DBA. (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 server option” on page 148.

Objects owned by dbo and SYS

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, any changes to permissions on system objects are *not* unloaded. You must grant or revoke these permissions as desired in the new database.

Unloading and replication

There are special considerations for unloading databases involved in replication. See “Unloading and reloading a database participating in replication” [*SQL Remote*] and “Upgrading SQL Remote” [*SQL Anywhere 10 - Changes and Upgrading*].

Unload Database wizard

The Unload Database wizard walks you through the steps for unloading a database in Sybase Central. When using this wizard to unload your database, you can choose to unload all the objects in a database, or a subset of tables from the database. Only tables for users selected in the Filter Objects by Owner dialog appear in the Unload Database wizard. If you want to view tables belonging to a particular database user, right-click the database you are unloading, choose Filter Objects by Owner from the popup menu, and then select the desired user in the resulting dialog.

The Unload Database wizard also gives you the option to reload into an existing database or a new database, rather than into a reload file.

♦ To unload a database file or a running database (Sybase Central)

1. From the Tools menu, choose SQL Anywhere 10 ► Unload Database.

   The Unload Database wizard appears.

2. Follow the instructions in the wizard.

Tip
You can also access the Unload Database wizard from within Sybase Central using any of the following methods:

- Selecting a database, and choosing Unload Database from the File menu.
- Right-clicking a database, and choosing Unload Database from the popup menu.

☞ For full information on unloading a database from Sybase Central, see “Exporting databases” [*SQL Anywhere Server - SQL Usage*].

Unload utility (dbunload)

Syntax

dbunload [*options*] [*directory*]

Option	Description
@data	Read options from the specified environment variable or configuration file.

Option	Description
-ac *"keyword=value; …"*	Supply connection parameters for the reload.
-an *database*	Create a database file with the same settings as the database being unloaded, and automatically reload it.
-ap *size*	Specify the page size of the new database (-an or -ar must also be specified).
-ar [*directory*]	Rebuild and replace the database.
-c *"keyword=value; …"*	Supply database connection parameters for unload.
-d	Unload data only.
-dc	Recalculate computed columns.
-e *table*, …	Do not unload listed tables (use with -d).
-ea *algorithm*	Specify which strong encryption algorithm to encrypt your database with: you can choose AES or AES_FIPS.
-ek *key*	Specify encryption key for new database (-an or -ar must also be specified).
-ep	Prompt for encryption key for the new database (-an or -ar must also be specified).
-er	Do not encrypt tables in the new database when unloading encrypted tables.
-et	Enable table encryption for the new database (-an or -ar must also be specified).
-g	Initialize materialized views during reload.
-ii	Perform an internal unload and internal reload (the default).
-ix	Perform an internal unload and external reload.
-k	Make auxiliary catalog for tracing.
-m	Do not preserve user IDs for a replicating database.
-n	Do not unload data—unload schema definition only.
-nl	Do not unload data—include LOAD/INPUT statements in reload script.
-o *filename*	Log output messages to a file.
-p *char*	Specify the escape character for external unloads (default "\").

Option	Description
-q	Run in quiet mode—do not display messages or windows.
-qc	Close the messages window on completion [Windows CE only].
-r *reload-file*	Specify the name and directory of generated reload command file (default *reload.sql*).
-t *table,…*	Unload only the listed tables (use with -d).
-u	Unordered data. Do not use indexes to unload data.
-v	Verbose messages.
-xi	Perform an external unload and internal reload.
-xx	Perform an external unload and external reload.
-y	Replace the command file without confirmation.

Description

The *directory* is the destination directory where the unloaded data is to be placed. The *reload.sql* command file is always relative to the current directory of the user.

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 details about supplying a file name and path in this mode, see "UNLOAD TABLE 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.

Exit codes are 0 (success) or non-zero (failure). See "Software component exit codes" [*SQL Anywhere Server - Programming*].

There are special considerations for unloading databases involved in replication. See "Unloading and reloading a database participating in replication" [*SQL Remote*].

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's 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.

☞ For more information about file names and paths for the Unload utility, see "UNLOAD TABLE 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;ENG=server-name -xx
```

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 of 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
```

This provides you with the opportunity to correct, remove, or alter the failing statement so that the rebuild can continue from the point at which it failed, which can save considerable time over restarting the rebuild.

When the *unprocessed.sql* file is generated, dbunload returns a failed error code to make other tools or scripts aware of the failed rebuild.

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 will 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=enc.db;START=dbeng10 -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 (dbeng10/dbsrv10) 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, as well as the encryption key of the newly-rebuilt database.

☞ For more information about encryption, see "-ep server option" on page 142, and "DatabaseKey connection parameter [DBKEY]" on page 217.

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.

Unload utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Connection parameters for reload database (-ac)

This option causes the Unload utility to connect to an existing database and reload the data directly into it, combining the operation of unloading a database and reloading the results into an existing database. This option is not supported on Windows CE.

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 (which should be 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 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.

Create a database for reloading (-an)

You can combine the operations of unloading a database, creating a new database, and loading the data using this option. This option is not supported on Windows CE.

Typically, you would use this option when you do not want to change the initialization options of your database. The options specified when you created the source database are used to create the new database.

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.

Set page size for the new database (-ap)

This option allows you to set the page size of the new database and 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. 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 server option" on page 150.

Rebuild and replace database (-ar)

This option creates a new database with the same settings as the old database, reloads it, and replaces the old database. 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 CE.

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 and the Replication Agent. The transaction log management is handled only if the database is used in replication: if there is no SQL Remote publisher or LTM check, then the old transaction log is not needed and is deleted instead of being copied to the specified directory.

☞ For more information on transaction log management, see "Backup methods for remote databases in replication installations" on page 775.

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.

Connection parameters for source database (-c)

This option specifies the connection parameters for the source database. For a description of the connection parameters, see "Connection parameters" on page 206. 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 the default location of *samples-dir*, see "The samples directory" on page 295.

Unload data only (-d)

With this option, none of the database definition commands are generated (CREATE TABLE, CREATE INDEX, and so on); *reload.sql* contains statements to reload the data only.

Recalculate computed columns (-dc)

Specifying this option causes 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);
```

Note

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*].

Exclude listed tables (-e)

Use this option to exclude the specified tables from the *reload.sql* file.

Note

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.

Specify encryption algorithm (-ea)

This option allows you to choose which strong encryption algorithm is used to encrypt your new database. You can choose either AES (the default) or AES_FIPS for the FIPS-approved algorithm. AES_FIPS uses a separate library. Algorithm names are case insensitive. If you specify the -ea option, you must also specify -ep or -ek. If you specify -ea without specifying -an, the -ea option is ignored.

☞ For more information about strong database encryption, see “Strong encryption” on page 873.

Separately licensed component required

ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.

See “Separately licensed components” [*SQL Anywhere 10 - Introduction*].

Specify encryption key (-ek)

This option allows you to specify an encryption key 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 AES or AES_FIPS as specified by the -ea option. If you specify the -ek option without specifying -ea, the algorithm used will be AES. If you specify -ek without specifying -an, the -ek option is ignored.

Caution

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.

☞ For more information about strong database encryption, see “Strong encryption” on page 873.

Prompt for encryption key (-ep)

This option prompts you to specify 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.

☞ For more information about strong database encryption, see “Strong encryption” on page 873.

Remove encryption from tables (-er)

Use the -er option to remove 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 will 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"
```

Enable table encryption (-et)

Use the -et option to enable 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.

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 will 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"
```

Initialize materialized views during reload (-g)

When you run the Unload utility, dbunload includes statements to recreate materialized view definitions in the reload script. By default, these views are left in the uninitialized state, that is, they are created in the database without any data. Once the database is reloaded, events that normally refresh materialized views eventually run, populating the materialized views with data. However, if you would rather have the materialized views initialized immediately, as part of the reload process, specify the -g option. Specifying -g calls the system procedure sa_refresh_materialized_views as the final step of the reload process. 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 will have to 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 “Initializing materialized views” [*SQL Anywhere Server - SQL Usage*].

Use internal unload, internal reload (-ii)

This option 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.

Use internal unload, external reload (-ix)

This option 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.

Make auxiliary catalog (-k)

Specifying this option 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.

☞ For more information about diagnostic tracing see "Advanced application profiling using diagnostic tracing" [*SQL Anywhere Server - SQL Usage*] and "sa_diagnostic_auxiliary_catalog table" [*SQL Anywhere Server - SQL Reference*].

Do not preserve user IDs (-m)
With this option, user IDs are not preserved for databases involved in replication.

Unload schema definition only (-n)
With this option, none of the data in the database is unloaded; *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.

Include LOAD/INPUT statements in the reload script (-nl)
This option is equivalent to using the -n (no data) option, except that the resulting *reload.sql* file 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 of the data for that table can be skipped using `dbunload -d -e` *table-name*.

Log output messages to file (-o)
Use this option to write output messages to the named file. The location of this file is relative to dbunload.

Escape character (-p)
Use this option to replace 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.

Operate quietly (-q)
Do not display output messages. 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.

Close messages window when dbunload completes (-qc)
By default, the dbunload messages window remains open until a user closes it. Specify this option if you want the messages window to close once the unload completes. This option is only available on Windows CE.

Specify reload filename (-r)
Use this option to modify 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.

Unload only listed tables (-t)

Use this option to specify 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.

Note

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.

Output unordered data (-u)
Normally, the data in each table is ordered by the primary key or clustered index if one is defined for the table. 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.

Enable verbose mode (-v)
The Unload utility displays the name of the table being unloaded, as well as the number of rows that have been unloaded. This option is available only when you run dbunload from a command prompt.

Use external unloading, internal reload (-xi)
This option 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.

Use external unloading, external reload (-xx)
This option 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.

Operate without confirming actions (-y)
Choosing this option replaces existing command files without prompting you 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.

The Upgrade utility

Note
In previous releases, you could upgrade a database to the newest version without performing an unload and reload. However, to upgrade a database to version 10, you must rebuild the database by performing an unload and reload. See "Upgrading SQL Anywhere" [*SQL Anywhere 10 - Changes and Upgrading*].

You can use the Upgrade utility to update the system tables and views, add new database options, and recreate all system stored procedures, as well as install jConnect support and change support for Java in the database.

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.

Back up before upgrading
As with any software, it is recommended that you make a backup of your database before upgrading.

Upgrade Database wizard

Note
The Upgrade Database wizard does not upgrade a version 9.0.2 or earlier database to version 10.0.0. To upgrade an existing database to version 10.0.0, you must unload and reload the database using dbunload or the Unload Database wizard. See "Upgrading SQL Anywhere" [*SQL Anywhere 10 - Changes and Upgrading*].

The Upgrade Database wizard helps you upgrade a version 10.0.0 database to a later version of the software from Sybase Central or to restore default settings such as database options, system tables and views, and system procedures.

♦ To upgrade a database (Sybase Central)

1. Connect to the database.

2. In the left pane, select the SQL Anywhere plug-in.
3. In the right pane, click the Utilities tab.
4. In the right pane, double-click Upgrade Database.

   The Upgrade Database wizard appears.
5. Follow the instructions in the wizard.

Tip
You can also access the Upgrade Database wizard by:

- Right-clicking the database, and choosing Upgrade Database from the popup menu.
- Selecting a database, and choosing Upgrade Database from the File menu.

Upgrade utility (dbupgrad)

Upgrade utility unsupported for upgrading to version 10
The Upgrade utility (dbupgrad) cannot be used to upgrade version 9.0.2 and earlier databases to version 10. To upgrade older databases to version 10, you must rebuild the database by performing an unload and reload. See "Upgrading SQL Anywhere" [*SQL Anywhere 10 - Changes and Upgrading*].

Syntax

dbupgrad [*options*]

Option	Description
@data	Read options from the specified environment variable or configuration file.
-c *"keyword=value; ..."*	Supply database connection parameters.
-i	Do not install Sybase jConnect support.
-o *filename*	Log output messages to a file.
-q	Run in quiet mode—do not display messages or windows.

Description

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.

You can also use the Upgrade utility to update the system tables and views, restore database options, and recreate all system stored procedures, as well as install jConnect support and change support for Java in the database.

Exit codes are 0 (success) or non-zero (failure).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

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.
For more information, see "Upgrading SQL Anywhere" [*SQL Anywhere 10 - Changes and Upgrading*].

Upgrade utility options

Specify environment variable or configuration file (@*data*)
Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Connection parameters (-c)
For a description of the connection parameters, see "Connection parameters" on page 206. The user ID must have DBA authority.

For example, the following command upgrades a database called sample10 and does not install jConnect support, connecting as user DBA with password sql:

```
dbupgrad -c "UID=DBA;PWD=sql;DBF=c:\sa10\sample10.db" -i
```

Do not install Sybase jConnect support (-i)
If you want to use the Sybase jConnect JDBC driver to access system catalog information, you need to install jConnect support. Use this option if you want to exclude the jConnect system objects. You can still use JDBC, as long as you do not access system information. If you want, you can add Sybase jConnect support later using Sybase Central or the ALTER DATABASE UPGRADE statement.

☞ For more information, see "Installing jConnect system objects into a database" [*SQL Anywhere Server - Programming*] and "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

Log output messages to file (-o)
Write output messages to the specified file.

Operate quietly (-q)
Do not display output messages.

The Validation utility

With the Validation utility, you can validate the indexes and keys on some or all of the tables and materialized views in the database. 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 have either DBA or VALIDATE authority. By default, dbvalid validates all the tables and materialized views in the database.

This utility can be used in combination with regular backups (see "Backup and Data Recovery" on page 761) to give you confidence in the integrity of the data in your database.

You can access the Validation utility in the following ways:

- From Sybase Central, using the Validate Database wizard.
- From Interactive SQL, using the VALIDATE statement.
- At a command prompt, using the dbvalid command. This is useful for incorporating into batch or command files.

Caution
Validating a table or an entire database should be performed while no connections are making changes to the database; otherwise, spurious errors may be reported indicating some form of database corruption even though no corruption actually exists.

Validate Database wizard

You can validate a database or individual tables from Sybase Central using the Validate Database wizard. To use this wizard, you have either DBA or VALIDATE authority.

The Validate Database wizard helps validate a database or table from Sybase Central.

Validation requires exclusive access to each table in turn. For this reason, it is best to validate when there is no other activity on the database.

♦ To validate a database (Sybase Central)

1. From the Tools menu, choose SQL Anywhere 10 ► Validate Database.

   The Validate Database wizard appears.

2. Follow the instructions in the 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 from the popup menu.
- Selecting a database, and choosing Validate Database from the File menu.

♦ To validate a running database (Sybase Central)

1. Connect to the database.
2. Right-click the database and choose Validate Database from the popup menu.

   The Validate Database wizard appears.
3. Follow the instructions in the wizard.

♦ To validate an individual table, materialized view, or index (Sybase Central)

1. Connect to the database.
2. Locate the table, materialized view, or index you want to validate.

   Note
   You can only validate materialized views that are enabled and initialized.
3. Right-click the table and choose Validate from the popup menu.

Validation utility (dbvalid)

Syntax

dbvalid [*options*] [*object-name*, …]

Option	Description
@data	Read options from the specified environment variable or configuration file.
-c *"keyword=value*; *…"*	Supply database connection parameters.
-d	Validate the database.
-fx	Validate tables or materialized views with express check.
-i	Validate the specified index.
-o *filename*	Log output messages to a file.

Option	Description
-q	Run in quiet mode—do not display messages.
-s	Validate database pages using checksums.
-t	Validate the specified tables or materialized views.
object-name	Specify the name of the table, materialized view, or index to validate.

Description

With the Validation utility, you can validate the indexes and keys on some or all of the tables and materialized views in the 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).

The Validation utility may return warnings about checksum violations for databases that do not have checksums enabled because the database server automatically calculates checksums for critical pages. See "Ensuring your database is valid" on page 776.

Validation requires exclusive access to each table in turn. 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).

☞ For more information about exit codes, see "Software component exit codes" [*SQL Anywhere Server - Programming*].

☞ For information on specific checks made during validation, see "VALIDATE statement" [*SQL Anywhere Server - SQL Reference*].

Validation utility options

Specify environment variable or configuration file (@*data*)

Use this option to read in options from the specified environment variable or configuration file.

☞ For more information about configuration files, see "Using configuration files" on page 587.

Connection parameters (-c)

For a description of the connection parameters, see "Connection parameters" on page 206. The user ID must have DBA authority or VALIDATE authority.

For example, the following command validates the database *c:\salesdata.db*, connecting as user DBA with password sql:

```
dbvalid -c "UID=DBA;PWD=sql;DBF=c:\salesdata.db"
```

Validate database (-d)

Use this option to validate that all table pages in the database belong to the correct object, and perform 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.

Express check for tables or materialized views (-fx)
Use this option to validate 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.

Validate specified indexes (-i)
Instead of validating tables, validate indexes. In this case, for dbvalid, each of the *object-name* values supplied represents an index rather than a table or materialized view, and has a name of the following form:

[[*owner*.]*table-name*.]*index-name*

Log output messages to file (-o)
Write output messages to the named file.

Operate quietly (-q)
Do not display output messages to the client. You can still log the messages to file using the -o option, however.

Validate checksums for a database (-s)
Checksums are used to determine whether a database page has been modified on disk. If you created a database with checksums enabled, you can validate the database using checksums. Checksum validation reads each page of the database from disk and calculates its 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 Server Messages window. The -s option cannot be used in conjunction with -d, -i, -t, or either of the -f options.

Validate tables and materialized views (-t)
The list of *object-name* values represents a list of tables and materialized views. This is the default behavior.

object-name The name of a table or materialized view to validate.

If -i is used, *object-name* refers to an index to validate instead.

Part IV. Monitoring Your Database

This section describes how to set up and configure the SQL Anywhere SNMP Extension Agent. It includes the OIDs supported by the SQL Anywhere SNMP Extension Agent and also lists the contents of the tables in the SQL Anywhere MIB and the RDBMS MIB.

CHAPTER 17

The SQL Anywhere SNMP Extension Agent

Contents

About this chapter

This chapter introduces you to the SQL Anywhere SNMP Extension Agent.

Introduction to the SQL Anywhere SNMP Extension Agent

If you are running SQL Anywhere on Windows XP/200x (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:

- **dbsnmp10.dll** The SQL Anywhere SNMP Extension Agent. This file is located in *install-dir \win32*.
- **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** The 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\win32*.

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 in some cases set, using the SQL Anywhere SNMP Extension Agent. See "The SQL Anywhere MIB" on page 713.
- **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 715.

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, as well as 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 information about the tables in the SQL Anywhere MIB, see "SQL Anywhere MIB Reference" on page 727.

☞ For information about setting values in the SQL Anywhere MIB, see "Setting values using the SQL Anywhere SNMP Extension Agent" on page 722.

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.
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 **""**.
1.3.6.1.4.897.2.4	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 (along with DBN if desired and DBKEY if 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 728.

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 8 of these virtual tables.

☞ For descriptions of the supported tables contained in the RDBMS MIB, see "RDBMS MIB Reference" on page 751.

The SQL Anywhere Extension Agent provides *read-only* access to all of 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.colnum.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.

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 2000 or Windows XP.

♦ To install SNMP

1. Open the Control Panel.

   ♦ On Windows 2000, from the Start menu choose Settings ► Control Panel.

   ♦ On Windows XP, from the Start menu choose Control Panel.

2. Double-click Add/Remove Programs.

   The Add/Remove Programs dialog appears.

3. In the left pane of the Add/Remove Programs dialog, click Add/Remove Windows Components.

   The Windows Components wizard appears.

4. On the first page of the Windows Components wizard, double-click Management and Monitoring Tools.

   The Management and Monitoring Tools dialog appears.

5. Select the Simple Network Management Protocol option and then click OK.

6. Click Next.

7. Click Finished when the installation completes.

8. If you are prompted, restart your computer.

Once you have installed SNMP on your computer, two services should be running on your computer: SNMP Service and SNMP Trap Service.

♦ To check that the SNMP services are installed on your computer

1. Open the Control Panel.

   ♦ On Windows 2000, from the Start menu choose Settings ► Control Panel.

   ♦ On Windows XP, from the Start menu choose Control Panel.

2. In the Control Panel, double-click Administrative Tools.

3. Double-click Services.

   The services SNMP Service and SNMP Trap Service appear in the list of local services.

If you installed SNMP before you installed SQL Anywhere, you need to stop and restart the SNMP service so it will detect the SQL Anywhere SNMP Extension Agent. If you installed SQL Anywhere and then installed SNMP, the SNMP service will detect the SQL Anywhere SNMP Extension Agent automatically.

♦ To restart the SNMP service (Control Panel)

1. Open the Control Panel.

   - On Windows 2000, from the Start menu choose Settings ► Control Panel.
   - On Windows XP, from the Start menu choose Control Panel.

2. In the Control Panel, double-click Administrative Tools.
3. Double-click Services.
4. Right-click SNMP Service and choose Restart from the popup menu.

   This stops and restarts the SNMP service.

♦ To restart the SNMP service (Command line)

1. Open a command prompt and execute the following command:

   ```
   net stop snmp
   ```

   This stops the SNMP service.

2. Execute 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\win32* 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 DB*n* 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 autostart a database server
  - 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.

☞ For more information about the rdbmsDbLimitedResourceTable table, see "rdbmsDbLimitedResourceTable" on page 754.

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 724.

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.

♦ To restart the SNMP service (Control Panel)

1. Open the Control Panel.

   - On Windows 2000, from the Start menu choose Settings ► Control Panel.
   - On Windows XP, from the Start menu choose Control Panel.

2. In the Control Panel, double-click Administrative Tools.

3. Double-click Services.

4. Right-click SNMP Service and choose Restart from the popup menu.

   This stops and restarts the SNMP service.

♦ To restart the SNMP service (Command line)

1. Open a command prompt and execute 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 608.

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;ENG=server1;DBN=sales;DBF=sales.db
Trap1=1.1.5 > 50000
UtilDbPwd=test
[DB2]
ConnStr=UID=DBA;PWD=sql;ENG=server1;DBN=field;DBF=field.db
UtilDbPwd=test
Disabled=1
[DB3]
ConnStr=UID=DBA;PWD=sql;LINKS=tcpip;ENG=server2;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 LINKS connection parameter is required to specify the protocol. 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 733.
- Database server statistics. See "SQL Anywhere MIB server statistics" on page 731.
- Database options. See "SQL Anywhere MIB database options" on page 743.
- Database properties. See "SQL Anywhere MIB database properties" on page 740.

- Database statistics. See "SQL Anywhere MIB database statistics" on page 737.

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
1.3.6.1.4.1.897.2.2.3.5.2	Database option ansi_integer_overflow on database 2	Off
1.3.6.1.4.1.897.2.3.1	Agent version	10.0.0(2459)
1.3.6.1.4.1.897.2.3.2.1	Connection string for database 1	UID=DBA;PWD=sql; ENG=server1; 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.opt = '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 the OIDs and information about the options and properties that can be set with the SQL Anywhere SNMP Extension Agent, see "The SQL Anywhere MIB" on page 728.

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. In order 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 713.

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.

In order 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 from the popup menu.

   The Computer Management dialog appears.

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 from the popup menu.

   The SNMP Service property sheet appears.

5. Click the Traps tab.

6. On the Traps tab, click Add.

   The SNMP Service Configuration dialog appears.

7. In the SNMP Service Configuration dialog, type **localhost** in the text box and then click Add.

8. Click OK to close the Service property sheet.

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 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, as well as 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 "Understanding database mirroring" on page 828.

The only other traps sent by the SQL Anywhere SNMP Extension Agent are dynamic traps. See "Creating dynamic traps" on page 724.

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:

```
Traptrapnum=[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 a list of OIDs in the SQL Anywhere MIB, see "SQL Anywhere MIB Reference" on page 727.

☞ For a list of OIDs in the RDBMS MIB, see "RDBMS MIB Reference" on page 751.

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 are 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 will not be 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.

You can specify as many Trap fields as you want in the *sasnmp.ini* file. The OID used for the trap will be `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 be 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 >= 51200, 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.

CHAPTER 18

SQL Anywhere MIB Reference

Contents

About this chapter

This chapter contains descriptions of the tables in the SQL Anywhere MIB.

The SQL Anywhere MIB

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 of the statistics, properties, and options that can be retrieved and set using the SQL Anywhere SNMP Extension Agent.

☞ For information about the SQL Anywhere SNMP Agent, see "Understanding SNMP" on page 713 .

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.

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 descriptions of the database server statistics, see "Server-level properties" on page 496 and "Database-level properties" on page 506.

OID	Type	Name	Statistic
1.3.6.1.4.1.897.2.1.1.1.*n*	Integer32	srvStatActiveReq	ActiveReq
1.3.6.1.4.1.897.2.1.1.2.*n*	Integer32	srvStatAvailIO	AvailIO
1.3.6.1.4.1.897.2.1.1.3.*n*	Counter64	srvStatBytesReceived	BytesReceived
1.3.6.1.4.1.897.2.1.1.4.*n*	Counter64	srvStatBytesReceivedUncomp	BytesReceivedUncomp
1.3.6.1.4.1.897.2.1.1.5.*n*	Counter64	srvStatBytesSent	BytesSent
1.3.6.1.4.1.897.2.1.1.6.*n*	Counter64	srvStatBytesSentUncomp	BytesSentUncomp
1.3.6.1.4.1.897.2.1.1.7.*n*	Counter64	srvStatCacheHitsEng	CacheHitsEng

OID	Type	Name	Statistic
1.3.6.1.4.1.897.2.1.1.8.*n*	Integer32	srvStatCachePinned	CachePinned
1.3.6.1.4.1.897.2.1.1.9.*n*	Counter64	srvStatCacheReadEng	CacheReadEng
1.3.6.1.4.1.897.2.1.1.10.*n*	Counter64	srvStatCacheReplacements	CacheReplacements
1.3.6.1.4.1.897.2.1.1.11.*n*	Integer32	srvStatCurrentCacheSize	CurrentCacheSize
1.3.6.1.4.1.897.2.1.1.12.*n*	Counter64	srvStatDiskReadEng	DiskReadEng
1.3.6.1.4.1.897.2.1.1.13.*n*	Integer32	srvStatFreeBuffers	FreeBuffers
1.3.6.1.4.1.897.2.1.1.14.*n*	Integer32	srvStatInternal	Internal
1.3.6.1.4.1.897.2.1.1.15.*n*	Integer32	srvStatUniqueClientAddresses	UniqueClientAddresses
1.3.6.1.4.1.897.2.1.1.16.*n*	Integer32	srvStatLockedHeapPages	LockedHeapPages
1.3.6.1.4.1.897.2.1.1.17.*n*	Counter64	srvStatMainHeapBytes	MainHeapBytes
1.3.6.1.4.1.897.2.1.1.18.*n*	Integer32	srvStatMainHeapPages	MainHeapPages
1.3.6.1.4.1.897.2.1.1.19.*n*	Integer32	srvStatMapPhysicalMemoryEng	MapPhysicalMemoryEng
1.3.6.1.4.1.897.2.1.1.20.*n*	Integer32	srvStatMaxCacheSize	MaxCacheSize
1.3.6.1.4.1.897.2.1.1.21.*n*	Integer32	srvStatMinCacheSize	MinCacheSize
1.3.6.1.4.1.897.2.1.1.22.*n*	Counter64	srvStatMultiPacketsReceived	MultiPacketsReceived
1.3.6.1.4.1.897.2.1.1.23.*n*	Counter64	srvStatMultiPacketsSent	MultiPacketsSent
1.3.6.1.4.1.897.2.1.1.24.*n*	Counter64	srvStatPacketsReceived	PacketsReceived
1.3.6.1.4.1.897.2.1.1.25.*n*	Counter64	srvStatPacketsReceivedUncomp	PacketsReceivedUncomp
1.3.6.1.4.1.897.2.1.1.26.*n*	Counter64	srvStatPacketsSent	PacketsSent
1.3.6.1.4.1.897.2.1.1.27.*n*	Counter64	srvStatPacketsSentUncomp	PacketsSentUncomp
1.3.6.1.4.1.897.2.1.1.28.*n*	Integer32	srvStatPeakCacheSize	PeakCacheSize
1.3.6.1.4.1.897.2.1.1.29.*n*	Integer32	srvStatRemoteputWait	RemoteputWait
1.3.6.1.4.1.897.2.1.1.30.*n*	Integer32	srvStatReq	Req
1.3.6.1.4.1.897.2.1.1.31.*n*	Counter64	srvStatSendFail	SendFail
1.3.6.1.4.1.897.2.1.1.32.*n*	Integer32	srvStatTotalBuffers	TotalBuffers

OID	Type	Name	Statistic
1.3.6.1.4.1.897.2.1.1.33.*n*	Integer32	srvStatUnschReq	UnschReq
1.3.6.1.4.1.897.2.1.1.34.*n*	Integer32	srvStatInternal	Internal
1.3.6.1.4.1.897.2.1.1.35.*n*	Integer32	srvStatCacheFile	CacheFile
1.3.6.1.4.1.897.2.1.1.36.*n*	Integer32	srvStatCacheFileDirty	CacheFileDirty
1.3.6.1.4.1.897.2.1.1.37.*n*	Integer32	srvStatCacheAllocated	CacheAllocated
1.3.6.1.4.1.897.2.1.1.38.*n*	Integer32	srvStatCachePanics	CachePanics
1.3.6.1.4.1.897.2.1.1.39.*n*	Integer32	srvStatCacheFree	CacheFree
1.3.6.1.4.1.897.2.1.1.40.*n*	Integer32	srvStatCacheScavenges	CacheScavenges
1.3.6.1.4.1.897.2.1.1.41.*n*	Integer32	srvStatCacheScavengeVisited	CacheScavengeVisited
1.3.6.1.4.1.897.2.1.1.42.*n*	Integer32	srvStatLockedCursorPages	LockedCursorPages
1.3.6.1.4.1.897.2.1.1.43.*n*	Integer32	srvStatQueryHeapPages	QueryHeapPages
1.3.6.1.4.1.897.2.1.1.44.*n*	Integer32	srvStatCarverHeapPages	CarverHeapPages
1.3.6.1.4.1.897.2.1.1.45.*n*	Integer32	srvStatHeapsRelocatable	HeapsRelocatable
1.3.6.1.4.1.897.2.1.1.46.*n*	Integer32	srvStatHeapsLocked	HeapsLocked
1.3.6.1.4.1.897.2.1.1.47.*n*	Integer32	srvStatHeapsQuery	HeapsQuery
1.3.6.1.4.1.897.2.1.1.48.*n*	Integer32	srvStatHeapsCarver	HeapsCarver
1.3.6.1.4.1.897.2.1.1.49.*n*	Integer32	srvStatMultiPageAllocs	MultiPageAllocs
1.3.6.1.4.1.897.2.1.1.50.*n*	Integer32	srvStatRequestsReceived	RequestsReceived
1.3.6.1.4.1.897.2.1.1.51.*n*	Integer32	srvStatExchangeTasks	ExchangeTasks

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 descriptions of database server properties, see "Database-level properties" on page 506.

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
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

OID	Type	Name	Property
1.3.6.1.4.1.897.2.1.2.27.*n*	String	srvPropNativeProcessorArchitecture	NativeProcessorArchitecture
1.3.6.1.4.1.897.2.1.2.28.*n*	String	srvPropNumProcessorsAvail	NumProcessorsAvail
1.3.6.1.4.1.897.2.1.2.29.*n*	String	srvPropInternal	Internal
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

OID	Type	Name	Property
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
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	srvPropNumLogicalProcessorsUsed	NumLogicalProcessorsUsed
1.3.6.1.4.1.897.2.1.2.75.*n*	String	srvPropNumPhysicalProcessorsUsed	NumPhysicalProcessorsUsed
1.3.6.1.4.1.897.2.1.2.76.*n*	String	srvPropDefaultNcharCollation	DefaultNcharCollation
1.3.6.1.4.1.897.2.1.2.77.*n*	String	srvPropCollectStatistics	CollectStatistics

OID	Type	Name	Property
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

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 descriptions of the database statistics, see "Server-level properties" on page 496 and "Database-level properties" on page 506.

OID	Type	Name	Property
1.3.6.1.4.1.897.2.2.1.1.*n*	Counter64	dbStatCacheHits	CacheHits
1.3.6.1.4.1.897.2.2.1.2.*n*	Integer32	dbStatCacheReadIndInt	CacheReadIndInt
1.3.6.1.4.1.897.2.2.1.3.*n*	Integer32	dbStatCacheReadIndLeaf	CacheReadIndLeaf
1.3.6.1.4.1.897.2.2.1.4.*n*	Counter64	dbStatCacheRead	CacheRead
1.3.6.1.4.1.897.2.2.1.5.*n*	Integer32	dbStatCacheReadTable	CacheReadTable
1.3.6.1.4.1.897.2.2.1.6.*n*	Integer32	dbStatChkpt	Chkpt
1.3.6.1.4.1.897.2.2.1.7.*n*	Integer32	dbStatChkptFlush	ChkptFlush
1.3.6.1.4.1.897.2.2.1.8.*n*	Integer32	dbStatChkptPage	ChkptPage
1.3.6.1.4.1.897.2.2.1.9.*n*	Integer32	dbStatCheckpointUrgency	CheckpointUrgency
1.3.6.1.4.1.897.2.2.1.10.*n*	Integer32	dbStatCheckpointLogBitmapSize	CheckpointLogBitmapSize
1.3.6.1.4.1.897.2.2.1.11.*n*	Integer32	dbStatCheckpointLogBitmapPagesWritten	CheckpointLogBitmapPagesWritten
1.3.6.1.4.1.897.2.2.1.12.*n*	Integer32	dbStatCheckpointLogCommitToDisk	CheckpointLogCommitToDisk

OID	Type	Name	Property
1.3.6.1.4.1.897.2.2.1.13.*n*	Integer32	dbStatCheckpointLogPageInUse	CheckpointLogPageInUse
1.3.6.1.4.1.897.2.2.1.14.*n*	Integer32	dbStatCheckpointLogPagesRelocated	CheckpointLogPagesRelocated
1.3.6.1.4.1.897.2.2.1.15.*n*	Integer32	dbStatCheckpointLogSavePreimage	CheckpointLogSavePreimage
1.3.6.1.4.1.897.2.2.1.16.*n*	Integer32	dbStatCheckpointLogSize	CheckpointLogSize
1.3.6.1.4.1.897.2.2.1.17.*n*	Integer32	dbStatCheckpointLogWrites	CheckpointLogWrites
1.3.6.1.4.1.897.2.2.1.18.*n*	Integer32	dbStatCheckpointLogPagesWritten	CheckpointLogPagesWritten
1.3.6.1.4.1.897.2.2.1.19.*n*	Integer32	dbStatCommitFile	CommitFile
1.3.6.1.4.1.897.2.2.1.20.*n*	Integer32	dbStatConnCount	ConnCount
1.3.6.1.4.1.897.2.2.1.21.*n*	Integer32	dbStatCurrIO	CurrIO
1.3.6.1.4.1.897.2.2.1.22.*n*	Integer32	dbStatCurrRead	CurrRead
1.3.6.1.4.1.897.2.2.1.23.*n*	Integer32	dbStatCurrWrite	CurrWrite
1.3.6.1.4.1.897.2.2.1.24.*n*	Integer32	dbStatDiskReadIndInt	DiskReadIndInt
1.3.6.1.4.1.897.2.2.1.25.*n*	Integer32	dbStatDiskReadIndLeaf	DiskReadIndLeaf
1.3.6.1.4.1.897.2.2.1.26.*n*	Counter64	dbStatDiskRead	DiskRead
1.3.6.1.4.1.897.2.2.1.27.*n*	Integer32	dbStatDiskReadTable	DiskReadTable
1.3.6.1.4.1.897.2.2.1.28.*n*	Counter64	dbStatDiskWrite	DiskWrite
1.3.6.1.4.1.897.2.2.1.29.*n*	Integer32	dbStatExtendDB	ExtendDB
1.3.6.1.4.1.897.2.2.1.30.*n*	Integer32	dbStatExtendTempWrite	ExtendTempWrite

OID	Type	Name	Property
1.3.6.1.4.1.897.2.2.1.31.*n*	Integer32	dbStatFullCompare	FullCompare
1.3.6.1.4.1.897.2.2.1.32.*n*	Integer32	dbStatGetData	GetData
1.3.6.1.4.1.897.2.2.1.33.*n*	Integer32	dbStatIdleCheck	IdleCheck
1.3.6.1.4.1.897.2.2.1.34.*n*	Integer32	dbStatIdleChkpt	IdleChkpt
1.3.6.1.4.1.897.2.2.1.35.*n*	Integer32	dbStatIdleChkTime	IdleChkTime
1.3.6.1.4.1.897.2.2.1.36.*n*	Integer32	dbStatIdleWrite	IdleWrite
1.3.6.1.4.1.897.2.2.1.37.*n*	Integer32	dbStatIndAdd	IndAdd
1.3.6.1.4.1.897.2.2.1.38.*n*	Integer32	dbStatIndLookup	IndLookup
1.3.6.1.4.1.897.2.2.1.39.*n*	Integer32	dbStatIOToRecover	IOToRecover
1.3.6.1.4.1.897.2.2.1.40.*n*	Integer32	dbStatInternal	Internal
1.3.6.1.4.1.897.2.2.1.41.*n*	Integer32	dbStatInternal	Internal
1.3.6.1.4.1.897.2.2.1.42.*n*	Integer32	dbStatLockTablePages	LockTablePages
1.3.6.1.4.1.897.2.2.1.43.*n*	Integer32	dbStatMapPages	MapPages
1.3.6.1.4.1.897.2.2.1.44.*n*	Integer32	dbStatMaxIO	MaxIO
1.3.6.1.4.1.897.2.2.1.45.*n*	Counter64	dbStatMaxRead	MaxRead
1.3.6.1.4.1.897.2.2.1.46.*n*	Integer32	dbStatMaxWrite	MaxWrite
1.3.6.1.4.1.897.2.2.1.47.*n*	Integer32	dbStatPageRelocations	PageRelocations
1.3.6.1.4.1.897.2.2.1.48.*n*	Integer32	dbStatProcedurePages	ProcedurePages

OID	Type	Name	Property
1.3.6.1.4.1.897.2.2.1.49.*n*	Integer32	dbStatQueryCachePages	QueryCachePages
1.3.6.1.4.1.897.2.2.1.50.*n*	Integer32	dbStatQueryLowMemoryStrategy	QueryLowMemoryStrategy
1.3.6.1.4.1.897.2.2.1.51.*n*	Counter64	dbStatQueryRowsMaterialized	QueryRowsMaterialized
1.3.6.1.4.1.897.2.2.1.52.*n*	Integer32	dbStatRecoveryUrgency	RecoveryUrgency
1.3.6.1.4.1.897.2.2.1.53.*n*	Integer32	dbStatLogFreeCommit	LogFreeCommit
1.3.6.1.4.1.897.2.2.1.54.*n*	Integer32	dbStatLogWrite	LogWrite
1.3.6.1.4.1.897.2.2.1.55.*n*	Integer32	dbStatRelocatableHeapPages	RelocatableHeapPages
1.3.6.1.4.1.897.2.2.1.56.*n*	Integer32	dbStatRollbackLogPages	RollbackLogPages
1.3.6.1.4.1.897.2.2.1.57.*n*	Integer32	dbStatTempTablePages	TempTablePages
1.3.6.1.4.1.897.2.2.1.58.*n*	Integer32	dbStatTriggerPages	TriggerPages
1.3.6.1.4.1.897.2.2.1.59.*n*	Integer32	dbStatViewPages	ViewPages
1.3.6.1.4.1.897.2.2.1.60.*n*	Integer32	dbStatVersionStorePages	VersionStorePages
1.3.6.1.4.1.897.2.2.1.61.*n*	Integer32	dbStatSnapshotCount	SnapshotCount
1.3.6.1.4.1.897.2.2.1.62.*n*	Integer32	dbStatLockCount	LockCount

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 descriptions of database properties, see "Database-level properties" on page 506.

OID	Type	Name	Property
1.3.6.1.4.1.897.2.2.2.1.*n*	String	dbPropAlias	Alias
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	dbPropInterna	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
1.3.6.1.4.1.897.2.2.2.26.*n*	String	dbPropInternal	Internal

OID	Type	Name	Property
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
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

OID	Type	Name	Property
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

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 descriptions of database options, see “Alphabetical list of options” on page 389.

OID	Type	Name	Option
1.3.6.1.4.1.897.2.2.3.1.*n*	String	dbOptAllowNullsByDefault*	allow_nulls_by_default

OID	Type	Name	Option
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	dbOptAnsiCloseCursorsOnRollback*	ansi_close_cursors_on_roll back
1.3.6.1.4.1.897.2.2.3.5.*n*	String	dbOptAnsiIntegerOverflow*	ansi_integer_overflow
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	dbOptAnsiUpdateConstraints*	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*	String	dbOptAutomaticTimestamp*	automatic_timestamp
1.3.6.1.4.1.897.2.2.3.11.*n*	String	dbOptBackgroundPriority*	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	dbOptBlockingTimeout*	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	dbOptCloseOnEndtrans*	close_on_endtrans
1.3.6.1.4.1.897.2.2.3.19.*n*	String	dbOptConnectionAuthentication*	connection_authentication
1.3.6.1.4.1.897.2.2.3.20.*n*	String	dbOptContinueAfterRaiseerror*	continue_after_raise_error
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

OID	Type	Name	Option
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*	String	dbOptFloatAsDouble*	float_as_double
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*	String	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.45.*n*	Integer32	dbOptInternal	Internal

OID	Type	Name	Option
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*	String	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
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*	String	dbOptOptimisticWaitForCommit*	optimistic_wait_for_commit

OID	Type	Name	Option
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*	String	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*	String	dbOptQueryPlanOnOpen*	query_plan_on_open
1.3.6.1.4.1.897.2.2.3.76.*n*	String	dbOptQuotedIdentifier*	quoted_identifier
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*	String	dbOptInternal	Internal
1.3.6.1.4.1.897.2.2.3.82.*n*	String	dbOptRITriggerTime*	ri_trigger_time
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

OID	Type	Name	Option
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
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*	String	dbOptTruncateWithAutocommit*	truncate_with_auto_commit
1.3.6.1.4.1.897.2.2.3.101.*n*	String	dbOptTsqlHexConstant*	tsql_hex_constant
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

OID	Type	Name	Option
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*	Integer32	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
1.3.6.1.4.1.897.2.2.3.112.*n*	String	dbOptIntegratedServerName*	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	dbOptWebserviceNamespaceHost*	webservice_namespace_host
1.3.6.1.4.1.897.2.2.3.116.*n*	Integer32	dbOptMaxQueryRequests*	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	dbOptSynchronizeMirrorOnCommit*	synchronize_mirror_on_commit
1.3.6.1.4.1.897.2.2.3.119.*n*	Integer32	dbOptHttpSessionTimeout*	http_session_timeout
1.3.6.1.4.1.897.2.2.3.120.*n*	String	dbOptUuidHasHyphens*	uuid_has_hyphens
1.3.6.1.4.1.897.2.2.3.121.*n*	String	dbOptAllowSnapshotIsolation*	allow_snapshot_isolation
1.3.6.1.4.1.897.2.2.3.122.*n*	String	dbOptVerifyPasswordFunction*	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	dbOptCollectStatisticsOnDmlUpdates*	collect_statistics_on_dml_updates
1.3.6.1.4.1.897.2.2.3.125.*n*	String	dbOptJavaMainUserid*	java_main_userid
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	dbOptSecureFeatureKey*	secure_feature_key
1.3.6.1.4.1.897.2.2.3.130.*n*	String	dbOptMaterializedViewOptimization*	materialized_view_optimization
1.3.6.1.4.1.897.2.2.3.131.*n*	Integer32	dbOptUpdatableStatementIsolation*	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	dbOptPostLoginProcedure*	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*	String	dbOptPercentAsComment*	percent_as_comment
1.3.6.1.4.1.897.2.2.3.136.*n*	String	dbOptJavaVmOptions*	java_vm_options

CHAPTER 19

RDBMS MIB Reference

Contents

About this chapter

This chapter contains descriptions of each of the tables in the RDBMS MIB.

The RDBMS MIB

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 10\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('CompanyName')
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	rdbmsDbInfoProductName	PROPERTY('ProductName')
1.3.6.1.2.1.39.1.2.1.2.*db*	String	rdbmsDbInfoVersion	PROPERTY('ProductVersion')

OID	Type	Name	Value returned
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.)
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 Extenstion 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.db*	String	rdbmsDbLimitedResourceName	Name of dbspace, or Transaction Log, Transaction Log Mirror, or Temporary File.
1.3.6.1.2.1.39.1.4.1.2.*n.db*	OID	rdbmsDbLimitedResourceID	`1.3.6.1.4.1.897.2`
1.3.6.1.2.1.39.1.4.1.3.*n.db*	Integer	rdbmsDbLimitedResourceLimit	Free space available on disk + current file size, in bytes
1.3.6.1.2.1.39.1.4.1.4.*n.db*	Integer	rdbmsDbLimitedResourceCurrent	Current file size
1.3.6.1.2.1.39.1.4.1.5.*n.db*	Integer	rdbmsDbLimitedResourceHighwater	Current size
1.3.6.1.2.1.39.1.4.1.6.*n.db*	Integer	rdbmsDbLimitedResourceFailure	0[1]
1.3.6.1.2.1.39.1.4.1.7.*n.db*	String	rdbmsDbLimitedResourceDescription	"Bytes"

[1] This OID is not supported by the SQL Anywhere SNMP Extenstion 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')

OID	Type	Name	Value returned
1.3.6.1.2.1.39.1.5.1.4.*db*	String	rdbmsSrvContact	PROPERTY('LicensedCompany')

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 Extenstion 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')

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 Extenstion Agent.

Part V. Maintaining Your Database

This section describes how to back up database files and how to use events and schedules to automate database administration.

CHAPTER 20

Backup and Data Recovery

Contents

About this chapter

This chapter describes how to protect your data against operating system crashes, file corruption, disk failures, and total computer failure.

The chapter describes how to make backups of your database, how to restore data from a backup, and how to run your server so that validation, performance, and data protection concerns are addressed.

Introduction to backup and recovery

A **backup** is a copy of the information in a database, held in some physically separate location from your database. If the database becomes unavailable, perhaps because of damage to a disk drive, you can **restore** it from the backup. Depending on the nature of the damage, it is often possible to restore from backups all committed changes to the database up to the time it became unavailable.

Recovery happens when the operating system or database server crashes, or the database server does not shut down properly. The database server checks on database startup whether the database was shut down cleanly at the end of the previous session. If it was not, the server executes an automatic recovery process to restore information. This mechanism recovers all changes up to the most recently committed transaction.

Questions and answers

To answer the question…	Consider reading…
What is a backup?	"Introduction to backup and recovery" on page 762
What is recovery?	"Introduction to backup and recovery" on page 762
What is a transaction log?	"The transaction log" on page 766
What are media and system failure?	"Protecting your data against failure" on page 763
From what kinds of failure do backups protect my data?	"Protecting your data against failure" on page 763
What tools are available for backups?	"Ways of making backups" on page 764
What types of backup are available?	"Types of backup" on page 769
What type of backup should I use?	"Designing backup procedures" on page 769
If my database file or transaction log becomes corrupt, what data may be lost?	"Protecting your data against media failure" on page 767
How are backups executed?	"Understanding backups" on page 766
How often do I perform backups?	"Scheduling backups" on page 770
Can I schedule automatic backups?	"Scheduling backups" on page 770
My database is involved in replication. How does this affect my backup strategy?	"A backup scheme for databases involved in replication" on page 773 "Backup methods for remote databases in replication installations" on page 775
How can I backup to tape?	"Backing up a database directly to tape" on page 796

To answer the question…	Consider reading…
How do I plan a backup schedule?	"Designing a backup and recovery plan" on page 776
Can I automate backups?	"Automating Tasks Using Schedules and Events" on page 809
How can I be sure that my database file is not corrupt?	"Ensuring your database is valid" on page 776 "Validating a database" on page 789
How can I be sure that my transaction log is not corrupt?	"Validating the transaction log on database startup" on page 786 "Validating a transaction log" on page 791
How can I run my database for maximum protection against failures?	"Configuring your database for data protection" on page 779
How can I ensure high availability and machine redundancy?	"Protecting against total computer failure" on page 780 "Making a live backup" on page 797
How do I perform a backup?	"Making a full backup" on page 788
How do I restore data from backups when a failure occurs?	"Recovering from media failure on the database file" on page 798 "Recovering from media failure on an unmirrored transaction log" on page 799 "Recovering from media failure on a mirrored transaction log" on page 800
How do I create a maintenance plan for a database?	"Creating a maintenance plan" on page 808

Protecting your data against failure

If your database has become unusable, you have experienced a database **failure**. SQL Anywhere provides protection against the following categories of failure:

Media failure The database file and/or the transaction log become unusable. This may occur because the file system or the device storing the database file becomes unusable, or it may be because of file corruption.

For example:

- The disk drive holding the database file or the transaction log file becomes unusable.
- The database file or the transaction log file becomes corrupted. This can happen because of hardware problems or software problems.

Backups protect your data against media failure.

☞ For more information, see “Understanding backups” on page 766.

System failure A system failure occurs when the computer or operating system goes down while there are partially completed transactions. This could occur when the computer is inappropriately turned off or restarted, when another application causes the operating system to crash, or because of a power failure.

For example:

- The computer or operating system becomes temporarily unavailable while there are partially completed transactions, perhaps because of a power failure or operating system crash, or because the computer is inappropriately restarted.

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.

☞ For more information about the recovery mechanism, see “Backup and recovery internals” on page 782.

☞ It is possible to recover uncommitted changes manually. For information, see “Recovering uncommitted operations” on page 802.

Ways of making backups

There are several distinct ways of making backups. This section introduces each of the major approaches, but does not address any issues of appropriate options.

You can make backups in the following ways:

- **Sybase Central** You can use the Backup Database and Create Backup Images wizards in Sybase Central to make a backup. You can access these wizards by selecting a database and choosing Backup Database or Create Backup Images from the File menu (or from the popup menu). See “Backing up a database directly to tape” on page 796.

  You can also use the Create Maintenance Plan wizard in Sybase Central to validate and back up your database on a regular schedule. See “Creating a maintenance plan” on page 808.

- **Backup utility** The dbbackup utility makes client-side backups. For example, executing the following command at the command prompt makes backup copies of the database and transaction log in the directory *c:\backup* on the client computer:

  ```
  dbbackup -c "connection-string" c:\backup
  ```

  You can make a backup copy of the database and transaction log in the directory *c:\backup* on the server computer by specifying the -s option:

  ```
  dbbackup -c "connection-string" -s c:\backup
  ```

  For more information, see “The Backup utility” on page 590.

- **SQL statement** You can use a SQL statement to make the database server execute a server-side backup operation. For example, the following statement places backup copies of the database file and transaction log into the directory *c:\backup* on the server computer.

  ```
  BACKUP DATABASE
  DIRECTORY 'c:\\backup';
  ```

  For more information, see "BACKUP statement" [*SQL Anywhere Server - SQL Reference*].

  You can also issue a BACKUP DATABASE statement using the dbbackup -s option. See "Backup utility (dbbackup)" on page 591.

- **Offline backup** The above examples are all online backups, executed against a running database. You can make offline backups by copying the database files when the database is not running.

  For more information, see "Types of backup" on page 769.

Notes

You must have BACKUP authority or REMOTE DBA authority to make online backups of a database.

Understanding backups

To understand what files you need to back up, and how you restore databases from backups, you need to understand how the changes made to the database are stored on disk.

The database file

When a database is shut 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 database option checkpoint_time
- When the estimated time to do a recovery operation exceeds the database option recovery_time
- When the database server is idle long enough to write all dirty pages
- When a connection issues a CHECKPOINT statement
- When the database server is running without a transaction log and a transaction is committed

Between checkpoints, you need both the database file and another file, called the transaction log, to ensure that you have a complete copy of all committed transactions.

☞ For more information about checkpoints, see "Checkpoints and the checkpoint log" on page 783, and "How the database server decides when to checkpoint" on page 786.

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 replication using SQL Remote, the Replication Agent, or database mirroring.

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 much 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 will be lost. However, if the database and transaction log are stored on different disks, then most, if not all, of the data can be recovered because you will still have either the full database or the transaction log (from which the database can be recovered) in the event of a disk failure.

☞ For more information on how to use a transaction log to protect against media failure, see “Protecting against media failure on the database file” on page 779.

Caution

The database file and the transaction log file must be located on the same physical computer as the database server. Database files and transaction log files located on a network drive can lead to poor performance, data corruption, and server instability.

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 happens:

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

In this way, 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 [database]” on page 405, and “delayed_commits option [database]” on page 411.

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 mirrored transaction log, 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 in the case of media failure on the transaction log. A mirrored transaction log also enables a database server to perform automatic validation of the transaction log on database startup. See “Protecting against media failure on the transaction log” on page 779.

Protecting your data against media failure

Backups protect your data against media failure.

☞ For an overview of data protection mechanisms, see “Protecting your data against failure” on page 763.

The practical aspects of recovery from media failure depend on whether the media failure is on the database file or the transaction log file.

Media failure on the database file If your database file is not usable, but your transaction log is still 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.

☞ For more information on how to configure your database system, see "Protecting against media failure on the database file" on page 779.

Media failure on the transaction log file Unless you use a mirrored transaction log, 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 mirrored transaction log 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.

☞ For more information, see "Protecting against media failure on the transaction log" on page 779.

Designing backup procedures

When you make a backup, you have a set of choices to make about how to manage transaction logs. The choices you make depend on a set of factors including the following:

- Is the database involved in replication?

  In this chapter, replication means SQL Remote replication, or MobiLink synchronization where dbmlsync is running, or a database using the Replication Agent. Each of these replication methods requires access to the transaction log, and potentially to old transaction logs.

- How fast is the transaction log file growing relative to your available disk space? If the transaction log is growing quickly, you may not be able to afford to keep transaction logs available.

Types of backup

This section assumes that you are familiar with basic concepts related to backups.

☞ For more information about concepts related to backups, see "Introduction to backup and recovery" on page 762, and "Understanding backups" on page 766.

Backups can be categorized in several ways:

- **Full backup and incremental backup** A **full backup** is a backup of both the database file and of the transaction log. An **incremental backup** is a backup of the transaction log only. Typically, full backups are interspersed with several incremental backups.

  For more information on making backups, see "Making a full backup" on page 788, and "Making an incremental backup" on page 789.

- **Server-side backup and client-side backup**
  To execute a server side backup, you execute the BACKUP statement or use the dbbackup -s option; the database server then performs the backup. You can easily build server side backup into applications because it is a SQL statement. Also, server-side backup is generally faster because the data does not have to be transported across the client/server communications system. See "BACKUP statement" [*SQL Anywhere Server - SQL Reference*].

  You can execute an online backup from a client computer using the Backup utility. See "Backup utility (dbbackup)" on page 591 and "DBBackup function" [*SQL Anywhere Server - Programming*].

  Instructions for server-side and client-side backups are given together for each backup procedure.

- **Archive backup and image backup**
  An **archive backup** copies the database file and the transaction log into a single archive file, 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, and you can only make full backups.

You should use an archive backup if you are backing up directly to tape. Otherwise, an image backup has more flexibility for transaction log file management.

For more information on archive backups, see "Backing up a database directly to tape" on page 796.

- **Online backup and offline backup**
  Backing up a running database provides a snapshot of a consistent database, even though other users are modifying the database. An offline backup consists simply of copying the files. You should only perform an offline backup when the database is not running, and when the database server was shut down properly.

  The information in this chapter focuses on online backups.

- **Live backup** A live backup is a *continuous* backup of the database that helps protect against total computer failure.

  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 in case the primary system running the database server becomes unusable. A live backup runs continuously, terminating only if the server shuts down. If you suffer a system failure, 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.

  For more information on when to use live backups, see "Protecting against total computer failure" on page 780.

  For more information on how to make a live backup, see "Making a live backup" on page 797.

Scheduling backups

Most backup schedules involve periodic full backups interspersed with incremental backups of the transaction log. There is no simple rule for deciding how often to make backups of your data. The frequency with which you make backups depends on the importance of your data, how often it is updated or changes, and other factors.

Most backup strategies involve occasional full backups, interspersed by several incremental backups. 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. Archive backups are always full backups.

The kinds of failure against which a backup schedule protects you depends not only on how often you make backups, but also on how you operate your database server. See "Configuring your database for data protection" on page 779.

You should always keep more than one full backup. *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.* You should also keep some of your full backups offsite 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.

☞ For more information on scheduling operations such as backups, see “Automating Tasks Using Schedules and Events” on page 809.

Making image backups

An image backup consists a copy of the database file and/or the transaction log, each as separate files. Image backups provide more flexibility for transaction log management than archive backups do.

You can make an image backup using the Backup utility or the BACKUP DATABASE statement. Image backups are available on all supported platforms, and are the only supported type of backup on NetWare and Windows CE.

Backing up a database to a tape drive

To make backups to tape using an image backup, you have to take each backup copy and put it on tape using a disk backup utility.

See also

- “BACKUP statement” [*SQL Anywhere Server - SQL Reference*]
- “Backup utility (dbbackup)” on page 591
- “RESTORE DATABASE statement” [*SQL Anywhere Server - SQL Reference*]
- “Making archive backups” on page 775

Understanding parallel database backups

When you perform a server-side image backup using the Backup utility (dbbackup) or 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 CE.

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. In most cases, the bottleneck is likely a physical disk, but it could also be any of the 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 not only provides a performance benefit from the I/O parallelism, but also improves the safety of the data in the event of a hardware failure.

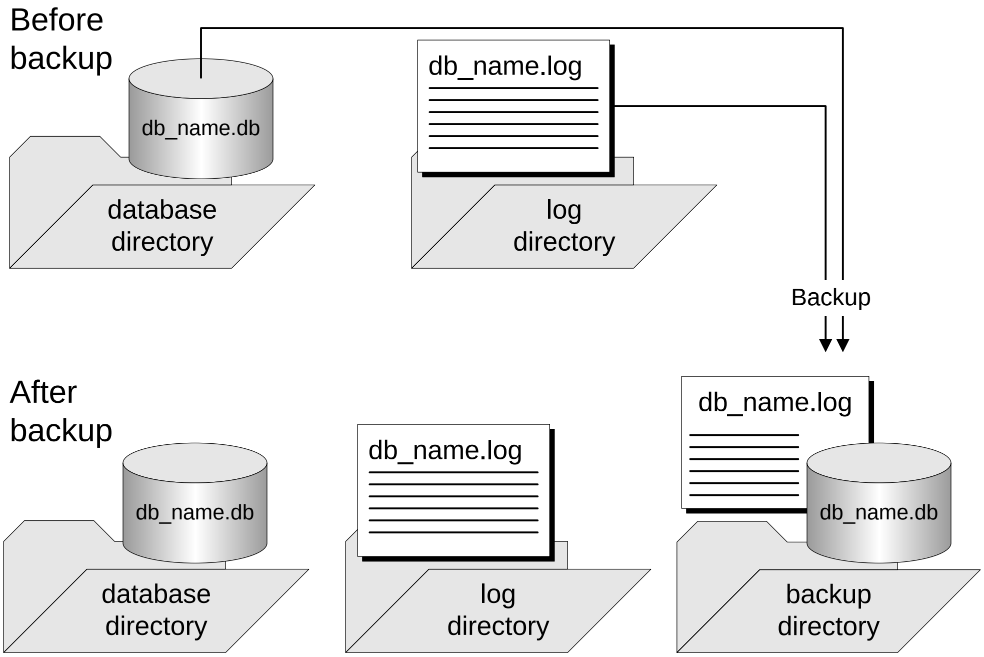
See also

- ♦ “BACKUP statement” [*SQL Anywhere Server - SQL Reference*]
- ♦ “Backup utility (dbbackup)” on page 591

A backup scheme for when disk space is plentiful

If disk space is not a problem on your production computer (where the database server is running) then you do not need to worry about choosing special options to manage the transaction log file or the checkpoint log. In this case, you can use an image backup. The simplest image backup makes copies of the database file and transaction log, and leaves the transaction log in place. All backups leave the database file in place.

A full backup of this kind is illustrated in the figure below. In an incremental backup, only the transaction log is backed up.

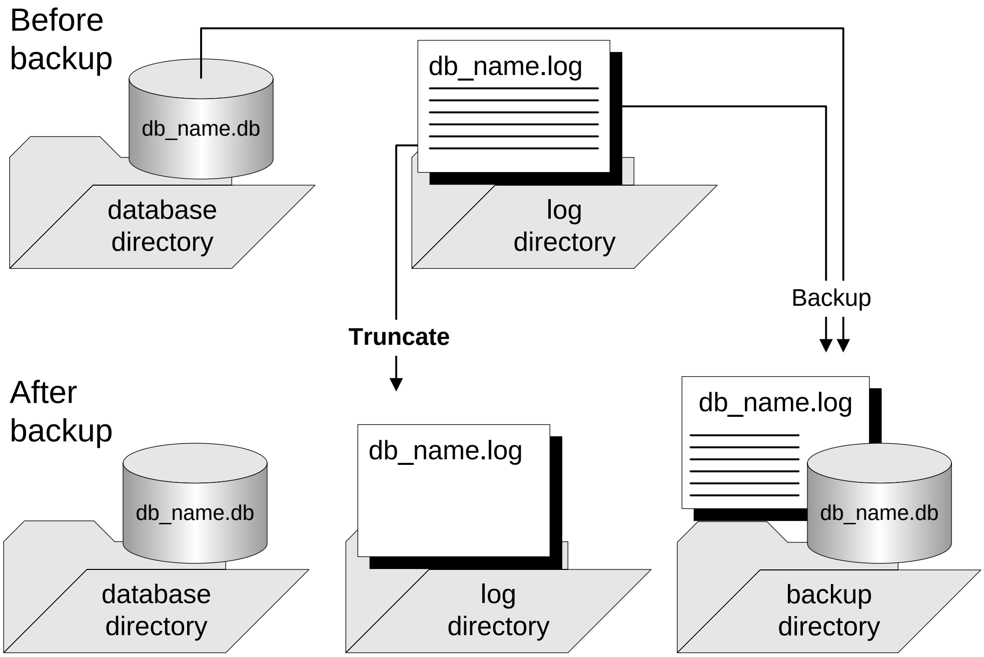


☞ For more information on how to perform backups of this type, see “Making a backup, continuing to use the original transaction log” on page 791.

A backup scheme for databases not involved in replication

In many circumstances, disk space limitations make it impractical to let the transaction log grow indefinitely. In this case, you can choose to delete the contents of the transaction log when the backup is complete, freeing the disk space. You should 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 as there may then be several different transaction logs since the last full backup. Each transaction log needs to be applied in sequence to bring the database up to date.

You can use this kind of backup at 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, as described in "A backup scheme for databases involved in replication" on page 773.

☞ For more information on how to perform a backup of this type, see "Making a backup, deleting the original transaction log" on page 792.

A backup scheme for databases involved in replication

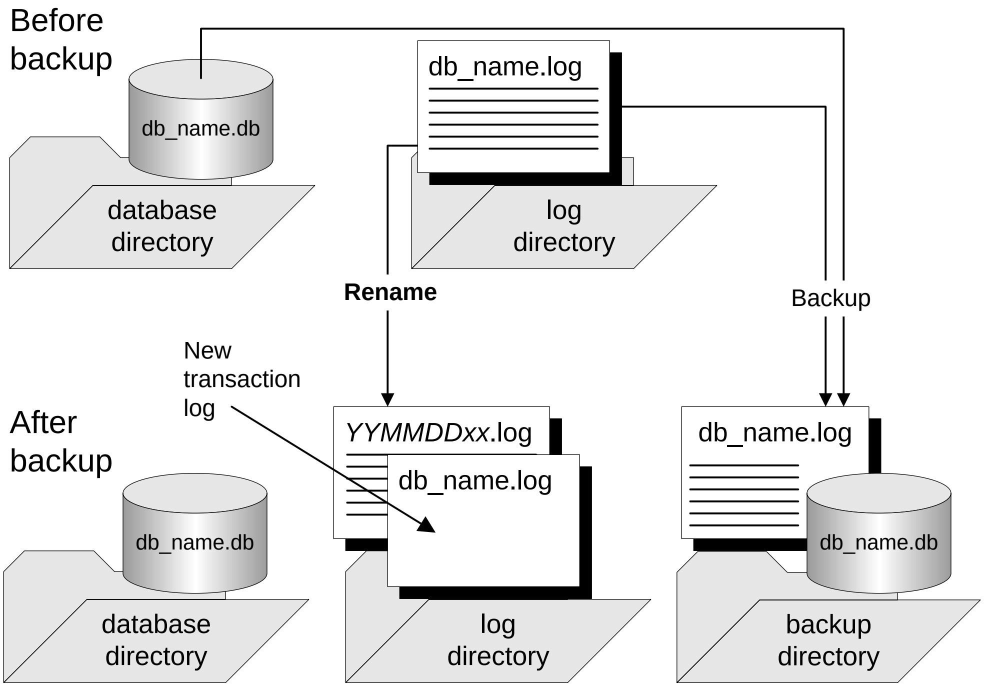
If your database is part of a SQL Remote installation, the Message Agent needs access to old transactions. If it is a consolidated database, it holds the master copy of the entire SQL Remote installation, and thorough backup procedures are essential.

If your database is a primary site in a Replication Server installation, the Replication Agent requires access to old transactions. However, disk space limitations often make it impractical to let the transaction log grow indefinitely.

If your database is participating in a MobiLink setup using dbmlsync, the same considerations apply. However, if your database is a MobiLink consolidated database, you do not need old transaction logs and can use a scheme for databases not involved in replication, as described in the previous section.

In these cases, 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 for the Message Agent and the Replication Agent.

This kind of backup is illustrated in the figure below.



☞ For more information on how to perform a backup of this kind, see "Making a backup, renaming the original transaction log" on page 793.

Offline transaction logs

In addition to backing up the transaction log, the backup operation renames 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 and the Replication 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 for distinguishability only, 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 and the Replication 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 and Replication Agent delete the offline files when they are no longer needed, saving disk space.

Backup methods for remote databases in replication installations

Backup procedures are not as crucial at remote databases as at the consolidated database. You may choose to rely on replication to the consolidated database as a data backup method. In the event of a 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 "The Log Translation utility" on page 637).

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 will be deleted automatically by the Message Agent when they are no longer needed.

Automatic transaction log renaming

You can 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.

Making archive backups

An archive backup is a single file that contains 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, and you can only make full 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.

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 XP/200x and Unix platforms only. On Windows CE, only image backups are permitted.

Backing up a database to a tape drive

You can perform direct backup to a tape drive using an archive backup. Archive backups are always full backups. An archive backup makes copies of both the database file and the transaction log, but these copies are placed into a single file.

See also

- “BACKUP statement” [*SQL Anywhere Server - SQL Reference*]
- “RESTORE DATABASE statement” [*SQL Anywhere Server - SQL Reference*]
- “Restoring an archive backup” on page 801
- “Making image backups” on page 771

Designing a backup and recovery plan

For reliable protection of your data, you should develop and implement a backup schedule. You should also ensure that you have a set of tested recovery instructions.

Typical schedules call for occasional full backups interspersed with several incremental backups. The frequency of each depends on the nature of the data that you are protecting.

If you use internal backups, you can use the scheduling features in SQL Anywhere to automate the task. Once you specify a schedule, the backups are performed automatically by the database server.

☞ For more information on automating backups, see “Automating Tasks Using Schedules and Events” on page 809.

The length of time your organization can function without access to the data in your database imposes a maximum recovery time, and you should develop and test a backup and recovery plan that meets this requirement.

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 mirrored transaction log.

☞ For more information about media failure, see “Protecting your data against media failure” on page 767.

Factors that affect recovery time

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 miscellaneous tasks that must be performed, such as entering recovery commands or retrieving and loading tapes.

Adding more files into the recovery scenario increases the places where recovery can fail. As the backup and recovery strategy develops, you should consider checking your recovery plan.

☞ For more information about how to implement a backup and recovery plan, see “Implementing a backup and recovery plan” on page 788.

Ensuring your database is valid

Database file corruption may not be apparent until the database server tries to access the affected part of the database. As part of your data protection plan, you should periodically check that your database has no errors.

You can do this by **validating** the database 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 permissions to perform validation activities.

Validation includes a scan of every row in every table and a look-up of each row in each index on the table. Validation requires exclusive access to each table in turn. For this reason, it is best to validate when there is no other activity on the database. Database validation does not validate data, continued row structure, or foreign key relationships if you perform an express validation using the -fx option.

☞ For information on using the Validate Database wizard, see "Validating a database" on page 789.

☞ For information on using the Validation utility (dbvalid), see "The Validation utility" on page 704.

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 the recovery steps. In this case, you can perform a validity check on the backup using the read-only database option. See "-r server option" on page 167.

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.

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. 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.

Validating checksums

If you created your database with checksums enabled, you can check the validity of the disk pages. 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) 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 Server Messages window.

☞ For more information about checksum validation, see “VALIDATE statement” [*SQL Anywhere Server - SQL Reference*], or “The Validation utility” on page 704.

Note
The database server calculates checksums for critical database pages in all databases, regardless of whether 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 checksums enabled, but that has a bad critical page, dbvalid can still return warnings about checksum violations.

See also

- “Validating a database” on page 789
- “Validating a transaction log” on page 791
- “Validating a single table” on page 791
- “VALIDATE statement” [*SQL Anywhere Server - SQL Reference*]
- “Improving performance when validating databases” on page 787

Configuring your database for data protection

There are several ways in which you can configure your database and the database server to provide protection against media failure while maintaining performance.

Protecting against media failure on the database file

When you create a database, by default the transaction log is put on the same device and in the same directory as the database. This arrangement does not protect against all kinds of media failure, and you should consider placing the transaction log in another location for production use.

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 result in improved performance by eliminating the need for disk head movement between the transaction log and the main database file.

You should not place the transaction log on a network directory. Reading and writing pages over a network gives poor performance and may result in file corruption.

☞ For more information on creating databases, see "Creating a database" [*SQL Anywhere Server - SQL Usage*].

☞ For more information on how to change the location of a transaction log, see "Changing the location of a transaction log" on page 805.

Protecting against media failure on the transaction log

It is recommended that you use a transaction log mirror when running high-volume or extremely 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 mirrored transaction log, 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 in the case of media failure on either log device; this purpose would be lost if the server continued with a single 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.

For more information, see "-fc server option" on page 143.

Where to store the transaction log mirror

There is a performance penalty for using a mirrored log as 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. Also, 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 mirrored transaction log are to use a disk controller that provides hardware mirroring, or operating-system level software mirroring, as provided by Windows XP/200x and NetWare. Generally, hardware mirroring is more expensive, but provides better performance.

See also

- Live backups provide additional protection that has some similarities to transaction log mirroring. For more information, see "Differences between live backups and transaction log mirrors" on page 780.
- For information on creating a database with a mirrored transaction log, see "The Initialization utility" on page 614.
- For information on changing an existing database to use a mirrored transaction log, see "The Transaction Log utility" on page 685.

Protecting against total computer failure

You can use a **live backup** to provide a redundant copy of the transaction log that is available for restart of your system on a secondary computer in case the computer running the database server becomes unusable.

A live backup runs continuously, terminating only if the server shuts down. If you suffer a system failure, 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.

☞ For more information on making a live backup, see "Making a live backup" on page 797.

☞ For information on restarting database using a live backup, see "Recovering from a live backup" on page 801.

An alternative to a live backup is to use database mirroring. See "Understanding database mirroring" on page 828.

Differences between live backups and transaction log mirrors

Both a live backup and a transaction log mirror to 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, 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 goes down.

- **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 mirrored transaction log contains all the information required for complete recovery of committed transactions. Depending on the load that the server is processing, the live backup may lag behind and may not contain all the committed transactions.

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.

☞ For more information about how to make a live backup, see “Making a live backup” on page 797.

Controlling transaction log size

The size of the transaction log can determine what kind of backup is right for you, and can also affect recovery times.

You can control how fast the transaction log file grows 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 sufficient.

Backup and recovery internals

This section describes the internal mechanisms used during backup and during automatic recovery mechanism from system failures.

Backup internals

When you issue a backup instruction, the database may be in use by many people. If you later need to use your backup to restore your database, you need to know what information has been backed up, and what has not.

The database server performs a backup as follows:

1. Issue a checkpoint. Further checkpoints are disallowed until the backup is complete.
2. Make a backup of the database files, if the backup instruction is for a full backup.
3. Make 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 instruction was issued.

   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. If the backup instruction requires the transaction log to be truncated or renamed, uncommitted transactions are carried forward to the new transaction log.

   ☞ For more information about renaming and truncating the transaction log, see “Designing backup procedures” on page 769.
5. Mark the backup image of the database to indicate that recovery is needed. This causes any operations that happened since the start of the backup to be applied. It also causes operations that were incomplete at the checkpoint to be undone if they were not committed.

Restrictions during backup and recovery

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 the one issued by the backup instruction itself.
- Any statement that causes a checkpoint. This includes data definition statements, as well as the LOAD TABLE and TRUNCATE TABLE statements.

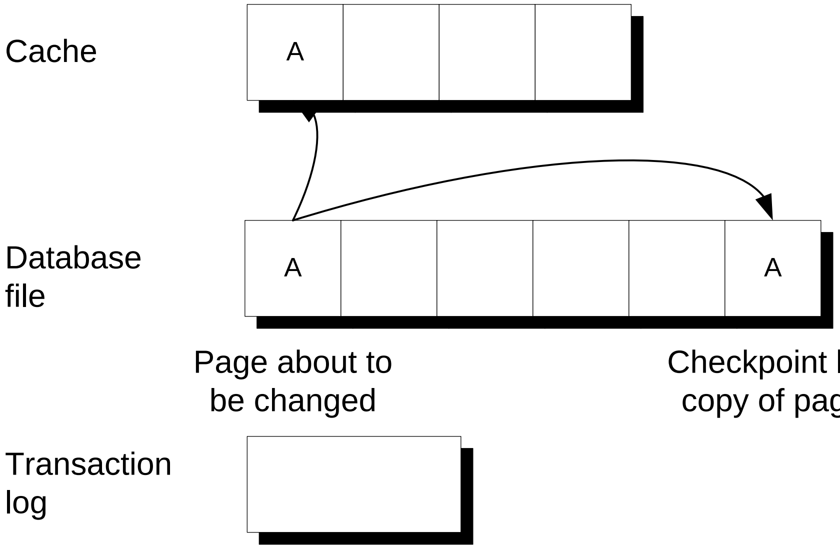
During recovery, including restoring backups, no action is permitted by other users of the database.

Checkpoints and the checkpoint log

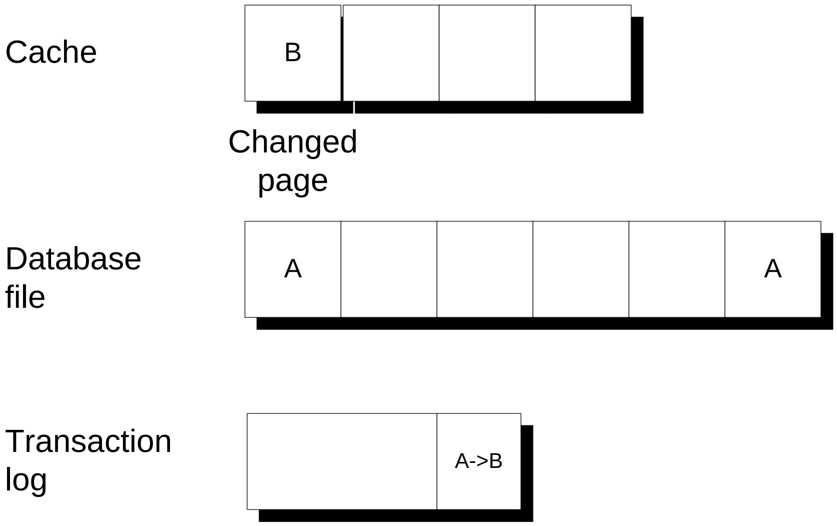
The database file is composed of pages: fixed size portions of hard disk. The checkpoint log is located at the end of the database file. Pages are added to the checkpoint log as necessary during a session, and the entire checkpoint log is deleted at the end of the session.

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

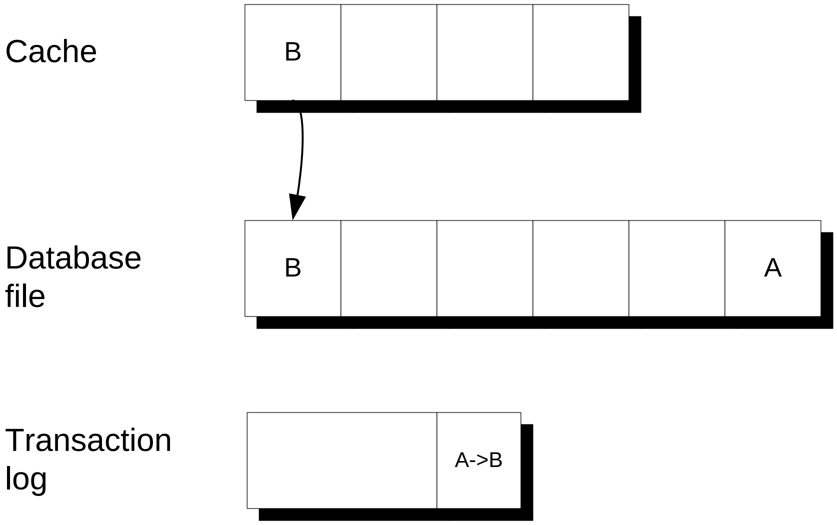
- It reads the page into memory, where it is held in the database cache.
- It makes a copy of the original page. These copied pages are the **checkpoint log**.



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.



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



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.

The entire checkpoint log, including all empty checkpoint log pages, is deleted at the end of each session. Deleting the checkpoint log causes the database to shrink in size.

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

☞ For more information about when checkpoints occur, see "How the database server decides when to checkpoint" on page 786.

Transactions and the rollback log

As changes are made to the contents of a database, a **rollback log** is kept for the purpose of canceling changes 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 rollback log contents 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**.

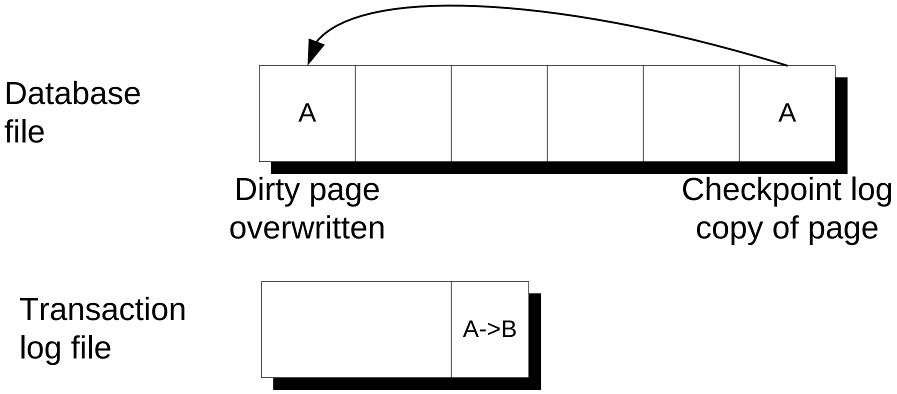
☞ For more information about transaction processing, see "Using Transactions and Isolation Levels" [*SQL Anywhere Server - SQL Usage*].

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** All pages are restored to their state at the most recent checkpoint by copying the checkpoint log pages over the changes made since the checkpoint.



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.

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 grows. 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. The server -gc option controls the maximum desired time, in minutes, between checkpoints. You can also set the desired time using the checkpoint_time option.

  For more information, see "-gc server option" on page 146.
- **Recovery Urgency** A heuristic to estimate the amount of time required to recover the database if it fails right now. The server -gr option controls the maximum desired time, in minutes, for recovery in the event of system failure. You can also set the desired time using the recovery_time option.

  For more information, see "-gr server option" on page 151.

The checkpoint and recovery urgencies are important only if the server does not have enough idle time to write dirty pages.

Frequent checkpoints make recovery quicker, but also create work for the server writing out dirty pages.

There are two database options that allow you to control the frequency of checkpoints. checkpoint_time controls the maximum desired time between checkpoints and recovery_time controls the maximum desired time for recovery in the event of system failure.

☞ For more information, see "checkpoint_time option [database]" on page 399 and "recovery_time option [database]" on page 449.

If, because of other activity in the database, the number of dirty pages falls to zero, and if the urgency is 50% or more, then a checkpoint takes place automatically since it is a convenient time.

Both the checkpoint urgency and recovery urgency values increase in value until the checkpoint occurs, at which point they drop to zero. They do not decrease otherwise.

Validating the transaction log on database startup

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.

If the database stopped because of a system failure, it is possible that some operations were 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 log and the mirror are different in the body of the shorter of the two, 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.

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 of 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*].

Express validation is only supported for databases created with Adaptive Server Anywhere 7.0 or later.

☞ For more information about the express check option, see the “VALIDATE statement” [*SQL Anywhere Server - SQL Reference*], “Validation utility (dbvalid)” on page 705, and the “sa_validate system procedure” [*SQL Anywhere Server - SQL Reference*].

Backup and recovery tasks

This section collects together instructions for tasks related to backup and recovery.

Implementing a backup and recovery plan

Regular backups and tested recovery commands are part of a comprehensive backup and recovery plan.

☞ For more information, see "Designing a backup and recovery plan" on page 776.

♦ To implement a backup and recovery plan

1. Create and verify your backup and recovery commands.
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 and identify any naming convention used, as well as the kind of backups 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.

☞ For more information about performing backups, see "Making a full backup" on page 788, and "Making an incremental backup" on page 789.

Making a full backup

A full backup is a backup of the database file and the transaction log file.

☞ For more information about the difference between a full backup and an incremental backup, see "Types of backup" on page 769.

♦ To make a full backup (overview)

1. Ensure that you have DBA authority on the database.
2. 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.

   ☞ For more information, see "Validating a database" on page 789.
3. Make a backup of your database file and transaction log.

   ☞ For information on how to perform the backup operation, see the following:

   ♦ "Making a backup, continuing to use the original transaction log" on page 791.

- “Making a backup, deleting the original transaction log” on page 792.
- “Making a backup, renaming the original transaction log” on page 793.

Notes

Validity checking requires exclusive access to entire tables on your database. For more information and alternative approaches, see “Ensuring your database is valid” on page 776.

If you validate your backup copy of the database, make sure that you do so in read-only mode. Start the database server with the –r option to use read-only mode.

Making an incremental backup

An incremental backup is a backup of the transaction log file only. Typically, you should make several incremental backups between each full backup.

☞ For more information about the difference between a full backup and an incremental backup, see “Types of backup” on page 769.

♦ To make an incremental backup (overview)

1. Ensure that you have DBA authority on the database.
2. Make a backup of your transaction log only, not your database file.

☞ For more information about how to perform the backup operation, see the following:

- “Making a backup, continuing to use the original transaction log” on page 791.
- “Making a backup, deleting the original transaction log” on page 792.
- “Making a backup, renaming the original transaction log” on page 793.

Notes

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.

☞ For more information about how to make a repeatable incremental backup command by renaming the backup copy of the transaction log, see “Renaming the backup copy of the transaction log during backup” on page 796.

Validating a database

Validating a database is a key part of the backup operation. You must have either DBA or VALIDATE authority to validate a database.

☞ For information about validating your database, see “Ensuring your database is valid” on page 776.

☞ For an overview of the backup operation, see “Making a full backup” on page 788.

Caution
Validating a table or an entire database should be performed while no connections are making changes to the database; otherwise, spurious 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. In the left pane of Sybase Central, select the database.
2. From the File menu, choose Validate Database

   The Validate Database wizard appears.
3. Follow the instructions in the wizard.

   A message box indicates whether the database is valid or not.

♦ To check the validity of an entire database (SQL)

- Execute the sa_validate stored procedure:

  ```
  CALL sa_validate;
  ```

  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"
  ```

  ☞ For more information, see “The Validation utility” on page 704.

Notes

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. You can only do this when there were no transactions in progress during the backup.

☞ For information on running databases in read-only mode, see “-r server option” on page 167.

Validating a single table

You can check the validity of a single table either from Sybase Central or using a SQL statement. You must have either DBA or VALIDATE authority to validate a table.

♦ To check the validity of a table (Sybase Central)

1. Open the Tables folder.
2. Right-click the table and choose Validate from the popup menu.

   A message box indicates whether the table is valid or not.

♦ 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 of 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. Another solution to errors reported by VALIDATE TABLE is to unload and reload your entire database. You should use the dbunload -u option so that it does not try to use a possibly corrupt index to order the data.

☞ For more information on using the REBUILD clause of the ALTER INDEX statement, see “ALTER INDEX statement” [*SQL Anywhere Server - SQL Reference*].

Validating a transaction log

You use the Log Translation utility (dbtran) to validate transaction logs, regardless of whether you have an online or offline transaction log. If the Log Translation utility can successfully read the log file, it is valid. See “The Log Translation utility” on page 637.

Making a backup, continuing to use the original transaction log

This task describes the simplest kind of backup, which leaves the transaction log untouched.

☞ For more information about when to use this type of backup, see “A backup scheme for when disk space is plentiful” on page 772.

♦ To make a backup, continuing to use the original transaction log (Sybase Central)

1. Start Sybase Central. Connect to the database as the DBA.
2. Right-click the database and choose Create Backup Images from the popup menu.

   The Create Backup Images wizard appears.

3. Click Next on the introductory page of the wizard.
4. Select the database that you want to back up.
5. On the next page, enter the name of a directory to hold the backup copies, and choose whether to perform a complete backup (all database files) or an incremental backup (transaction log file only).
6. On the next page, select the option Continue To Use The Same Transaction Log.
7. Click Finish to start the backup.

The procedure describes a client-side backup. There are more options available for this kind of backup.

If you choose a server-side backup, and the server is running on a different computer from Sybase Central, you cannot use the Browse button to locate a directory in which to place the backups. The Browse button browses the client computer, while the backup directory is relative to the server.

♦ To make a backup, continuing to use the original transaction log (SQL)

- If you are using the BACKUP statement, use the following clauses only:

  ```
  BACKUP DATABASE
  DIRECTORY directory-name
  [ TRANSACTION LOG ONLY ];
  ```

  Include the TRANSACTION LOG ONLY clause if you are making an incremental backup.

♦ 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.

Making a backup, deleting the original transaction log

If your database is not involved in replication, and if you have limited disk space on your online computer, you can delete the contents of the online transaction log (**truncate** the log) when you make a backup. In this case, you need to use every backup copy made since the last full backup during recovery from media failure on the database file.

☞ For more information about when to use this type of backup, see "A backup scheme for databases not involved in replication" on page 772.

♦ To make a backup, deleting the transaction log (Sybase Central)

1. Start Sybase Central. Connect to the database as the DBA.
2. Right-click the database and choose Create Backup Images from the popup menu.

   The Create Backup Images wizard appears.

3. Click Next on the introductory page of the wizard.

4. Select the database that you want to back up.

5. On the next page, enter the name of a directory to hold the backup copies, and choose whether to perform a complete backup (all database files) or an incremental backup (transaction log file only).

6. On the next page, select the option Truncate the Transaction Log.

7. Click Finish to start the backup.

♦ 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)

- From the command prompt, enter 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.

Notes

Before the online transaction log can be erased, all outstanding transactions must terminate. If there are outstanding transactions, your backup cannot complete.

☞ For information, see “Backup internals” on page 782.

You can determine which connection has an outstanding transaction using the sa_conn_info system procedure. If necessary, you can disconnect the user with a DROP CONNECTION statement.

☞ For more information, see “Determining which connection has an outstanding transaction” on page 795.

Making a backup, renaming the original transaction log

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.

☞ For more information about when to use this set of backup options, see "A backup scheme for databases involved in replication" on page 773.

♦ To make a backup, renaming the transaction log (Sybase Central)

1. Start Sybase Central. Connect to the database as the DBA.
2. Right-click the database and choose Create Backup Images from the popup menu.

   The Create Backup Images wizard appears.
3. Click Next on the introductory page of the wizard.
4. Select the database that you want to back up.
5. On the next page, enter the name of a directory to hold the backup copies, and choose whether to perform a complete backup (all database files) or an incremental backup (transaction log file only).
6. On the next page, select the option Rename the Transaction Log.
7. Click Finish to start the backup.

♦ 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)

- From a command prompt, enter the following command. You must enter the command on a single 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.

Notes

Before the online transaction log can be renamed, all outstanding transactions must terminate. If there are outstanding transactions, your backup will not complete.

☞ For information, see "Backup internals" on page 782.

You can determine which connection has an outstanding transaction using the sa_conn_info system procedure. If necessary, you can disconnect the user with a DROP CONNECTION statement.

☞ For more information, see “Determining which connection has an outstanding transaction” on page 795.

Determining which connection has an outstanding transaction

If you are performing a backup that renames or deletes the transaction log on a database created with version 8.0 or later of Adaptive Server Anywhere, incomplete transactions are carried forward to the new transaction log.

If you are performing a backup that renames or deletes the transaction log on a database created with version 7.x or earlier of Adaptive Server Anywhere, however, and if there are outstanding transactions, the backup must wait until those transactions are complete before it can complete.

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 connections.

♦ To determine which connection has an outstanding transaction (SQL)

1. Connect to the database from Interactive SQL or another application that can call stored procedures.

2. Execute the sa_conn_info system procedure:

   ```
   CALL sa_conn_info
   ```

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

   ☞ For more information, 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 user ID DBA and password sql:

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

   ☞ For more information, see “The SQL Anywhere Console utility” on page 668.

2. Double-click each connection, and inspect the Uncommitted Ops entry to see which users have uncommitted operations. If necessary, you can disconnect the user to enable the backup to finish.

Renaming 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.

You can make a repeatable incremental backup command by renaming 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'
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 for distinguishability only, not for ordering.

Backing up a database directly to tape

An archive backup makes a copy of the database file and transaction log file in a single archive destination. Only server-side, full backups can be made in this manner. When you make an archive backup in Sybase Central, you have the option of backing up the database directly to tape or to disk.

An extension of *.1* is added to the file name you specify in the Backup Database wizard or BACKUP statement.

☞ For more information, see "Types of backup" on page 769.

♦ To make an archive backup to tape (Sybase Central)

1. Start Sybase Central. Connect to the database as the DBA.
2. Right-click the database and choose Backup Database from the popup menu.

   The Backup Database wizard appears.

3. Click Next on the introductory page of the wizard.
4. Select the database that you want to back up to tape.
5. On the second page of the wizard, select the On Tape to the Following Device option and enter the tape drive in the text box below.
6. Click Finish to start the 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, such as replacing the tape, when there is no more space on the backup archive device.

Notes

The BACKUP statement makes an entry in the text file *backup.syb*, in the same directory as the server executable.

☞ For information about restoring from an archive backup, see "Restoring an archive backup" on page 801.

Example

The following statement makes a 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.

The following statement makes an archive file on disk named *c:\backup\archive.1*.

```
BACKUP DATABASE
TO 'c:\\backup\\archive';
```

Making 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 in case the primary system running the database server becomes unusable. A live backup runs continuously, terminating only if the server shuts down. If you suffer a system failure, 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 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.

☞ For more information about live backups, see “Protecting against total computer failure” on page 780.

You carry out a live backup of the transaction log by using the dbbackup utility with the -l option.

♦ To make a live backup

1. Set up a secondary computer from which you can run the database if the online computer fails. For example, 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;ENG=testsrv;DBN=test;LINKS=tcpip" c:\backup
   ```
3. Run a live backup of the transaction log to the secondary computer.

   ```
   dbbackup -l path\file-name.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.

Recovering from media failure on the database file

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.

When you have multiple transaction logs, it is possible that transactions may span 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.

☞ For more information, see “Recovering from multiple transaction logs” on page 803.

☞ For more information about the backup types discussed here, see “Designing backup procedures” on page 769.

♦ To recover from media failure on the database file

1. Make an extra backup copy of the current transaction log. The database file is gone, and the only record of changes since the last backup is in the transaction log.
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. You can use either of the following methods to do this.

   If you want to apply each transaction log manually, for each log file, in chronological order 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:

      ```
      dbeng10 database-name.db -a log-name.log
      ```

      The database server shuts down automatically once the transactions are applied.

   c. Once you have applied all of 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.

      ```
      dbeng10 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:

      ```
      dbeng10 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.

   ☞ For more information, see “Validating a database” on page 789.

6. Make a post-recovery backup.

7. Move the database file to the production directory.

8. Allow users to access the production database.

Recovering from media failure on an unmirrored transaction log

If your database is a primary site in a Replication Server installation, or a consolidated database in a SQL Remote installation, you should use a mirrored transaction log, or hardware equivalent.

☞ For more information, see “Protecting against media failure on the transaction log” on page 779.

♦ To recover from media failure on an unmirrored transaction log (partial recovery)

1. Make an extra backup copy of the database file immediately. With the transaction log gone, the only record of the changes between the last backup and the most recent checkpoint is in the database file.

2. Delete or rename the transaction log file.

3. Restart the database with the -f option.

   ```
   dbeng10 samples-dir\demo.db -f
   ```

   > **Caution**
   > This command should only be used when the database is not participating in a SQL Remote or Replication Server replication 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 server reports the lack of a transaction log as an error. With the option, the 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.

   ☞ For information about the default location of *samples-dir*, see “The samples directory” on page 295.

Recovering from media failure on a mirrored transaction log

The following steps explain how to recover from a media failure when you are using a mirrored transaction log. If your database is a primary site in a Replication Server installation, or a consolidated database in a SQL Remote installation, you should use a mirrored transaction log, or hardware equivalent.

♦ To recover from media failure on a mirrored transaction log

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 on the transaction log and on its mirror to see which one generates an error message. 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 so that you have two identical files again.

4. Restart the server.

Recovering 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 "Protecting against total computer failure" on page 780.

♦ 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:

   ```
   dbeng10 samples-dir\demo.db -ad directory-name
   ```

   ☞ For information about the default location of *samples-dir*, see "The samples directory" on page 295.

   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.

Restoring an archive backup

If you use an archive backup (typically to tape), you use the RESTORE statement to recover your data.

☞ For more information about making archive backups, see "Backing up a database directly to tape" on page 796.

♦ To restore a database from an archive backup (Sybase Central)

1. In Sybase Central, connect to a database as the DBA.

2. From the Tools menu, choose SQL Anywhere 10 ► Restore Database.

   The Restore Database wizard appears.

3. Follow the instructions in the wizard.

♦ To restore a database from an archive backup (Interactive SQL)

1. Start a personal database server. Use a command such as the following, which starts a server named restore:

```
dbeng10 -n restore
```

2. Start Interactive SQL. On the Identification tab of the Connect dialog, enter a user ID of **DBA** and a password of **sql**. Leave all other fields on this tab blank.
3. Click the Database tab and enter a database name of **utility_db**. Leave all other fields on this tab blank.
4. Click OK to connect.
5. Execute the RESTORE statement, specifying the archive root. At this time, you can choose to restore an archived database to its original location (default), or to a different computer with different device names using the RENAME clause.

   ☞ For more information, 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';
```

☞ For more information, see "RESTORE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

Recovering 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.

♦ To recover uncommitted operations from a transaction log (Sybase Central)

1. In Sybase Central, choose Tools ► SQL Anywhere 10 ► Translate Log File.

   The Translate Log File wizard appears.
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 on the Log Translation utility, see “The Log Translation utility” on page 637.

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.

Recovering from multiple transaction logs

SQL Anywhere allows transactions to span multiple transaction log files. If a database is backed up part way through a transaction, the transaction may span two transaction log files. When this occurs, the first part of the transaction is contained in the offline transaction log, while the second part of the transaction is contained in the online transaction log.

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* in case there are transactions that span multiple transaction logs. If the transaction logs are not applied in the correct order, then portions of transactions that span multiple transaction logs are rolled back.

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 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.

- Use the -ad server option when starting the database 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.

- Use the -ar server option when 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.

- 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.

☞ For more information, see “The Transaction Log utility” on page 685.

Recovering from multiple transaction logs using the -a 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 log 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.

☞ For more information, see "-ad database option" on page 195.

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*:

   ```
   dbeng10 backupdemo.db -ad c:\backuplogs
   ```

   The database server applies the transaction logs to the backup copy of the database and then shuts down.

Recovering from 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—it will not continue to run. 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.
2. Start the database server and apply the current transaction log to the backup copy of your database.

☞ For more information, see "-a database option" on page 194.

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*:

   ```
   dbeng10 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:

```
dbeng10 backupdemo.db -a demo.log
```

The database server applies the current transaction log to the backup copy of the database and then shuts down.

Recovering from 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 to generate a file (named by -n) containing all the transactions from the logs in the specified directory.

You need to use -m because if you translate each log individually using dbtran, any transactions that span transaction log files could be rolled back. This is because when dbtran translates a log, it adds a ROLLBACK statement to the end of the log to undo any uncommitted transactions. In cases where 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 your transactions are translated.

☞ For more information, see "The Transaction Log utility" on page 685.

♦ To recover from multiple transaction logs using the 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*:

   ```
   dbeng10 backupdemo.db
   ```

3. Apply the *recoverylog.sql* file to the database from Interactive SQL:

   ```
   dbisql "UID=DBA;PWD=sql;END=backupdemo" READ recoverylog.sql
   ```

Changing the location of a transaction log

The database must not be running when you change the location of a transaction log.

☞ For more information about how to choose the location of a transaction log, see "Protecting against media failure on the database file" on page 779.

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

1. From the Tools menu, choose SQL Anywhere 10 ▶ Change Log File Settings.
2. Follow the instructions in the 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. Enter the following command at the command prompt:

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

   ☞ For more information about dblog options, see "Transaction log utility options" on page 686.

Creating 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 either 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 "Protecting against media failure on the transaction log" on page 779.

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

1. In Sybase Central, choose Tools ▶ SQL Anywhere 10 ▶ Create Database.
2. Follow the instructions in the 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';
  ```

  ☞ For more information, 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
  ```

  ☞ For more information about initialization options, see “Initialization utility options” on page 616.

Starting a transaction log mirror for an existing database

Using the Transaction Log utility, you can choose to maintain a transaction log mirror for an existing database any time the database is not running. This option is available from either Sybase Central or the dblog utility.

☞ For more information about why you may want to use a transaction log mirror, see “Protecting against media failure on the transaction log” on page 779.

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

1. From the Tools menu, choose SQL Anywhere 10 ► Change Log File Settings.
2. Follow the instructions in the wizard.

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

1. Ensure that the database is not running.
2. Enter the following command at the command prompt:

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

   ☞ For more information about dblog options, see “Transaction log utility options” on page 686.

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

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 on a database:

- validating the database
- backing up the database
- managing the maintenance log

You create a maintenance plan from Sybase Central using the Create Maintenance Plan wizard. Each time the maintenance plan runs, a maintenance report is saved in the database. You can view this report from Sybase Central, or you can have the maintenance log emailed to you after each time the maintenance plan executes on the database.

♦ To create a maintenance plan

1. Connect to your database from Sybase Central.
2. Open the Maintenance Plans folder for the database.
3. From the File menu, choose New ▶ Maintenance Plan.

   The Create Maintenance Plan wizard appears.
4. Follow the instructions in the wizard.

   For information about the available settings, see:

   - "Defining schedules" on page 813
   - "VALIDATE statement" [*SQL Anywhere Server - SQL Reference*]
   - "Types of backup" on page 769
   - "xp_startsmtp system procedure" [*SQL Anywhere Server - SQL Reference*]

♦ To view the maintenance report

1. Once the maintenance plan has been executed, connect to your database from Sybase Central.
2. Open the Maintenance Plans folder.
3. Double-click your maintenance plan.
4. Double-click the report on the Reports tab in the right pane.

   The Maintenance Plan property sheet appears. The Details pane contains the log for the maintenance plan.

CHAPTER 21

Automating Tasks Using Schedules and Events

Contents

About this chapter

This chapter describes how to use scheduling and event handling features of SQL Anywhere to automate database administration and other tasks.

Introduction

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**.

Chapter contents

This chapter contains the following material:

- An introduction to scheduling and event handling (this section).
- Concepts and background information to help you design and use schedules and event handlers:
  - "Understanding schedules" on page 813
  - "Understanding system events" on page 815
- A discussion of techniques for developing event handlers:
  - "Developing event handlers" on page 819
- Internals information:
  - "Schedule and event internals" on page 821
- Step by step instructions for how to perform automation tasks.
  - "Event handling tasks" on page 823

Questions and answers

To answer the question…	Consider reading…
What is a schedule?	"Understanding schedules" on page 813.
What is an event?	"Understanding events" on page 812
What is a system event?	"Understanding system events" on page 815
What is an event handler?	"Understanding event handlers" on page 819
How do I debug event handlers?	"Developing event handlers" on page 819
How does the database server use schedules to trigger event handlers?	"How the database server checks for scheduled events" on page 821

To answer the question…	**Consider reading…**
How can I schedule regular backups?	"Understanding schedules" on page 813.
What kind of system events can the database server use to trigger event handlers?	"Understanding system events" on page 815. "CREATE EVENT statement" [*SQL Anywhere Server - SQL Reference*].
What connection do event handlers get executed on?	"How event handlers are executed" on page 822.
How do event handlers get information about what triggered them?	"Developing event handlers" on page 819 "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.

  For more information, see “Understanding schedules” on page 813.

- **System events** are associated with a particular type of condition that is tracked by the database server.

  For more information, see “Understanding system events” on page 815.

- **Manual events** are fired explicitly using the TRIGGER EVENT statement.

  For more information, “Triggering an event handler” on page 824.

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.

The following examples give some ideas for scheduled actions that may be useful.

☞ For more information, see "CREATE EVENT statement" [*SQL Anywhere Server - SQL Reference*].

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;
```

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 have a recurrence. 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. Connect to your database as a user with DBA authority.
2. Double-click the Events folder for your database.
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.

   The Create Schedule wizard appears.
6. Follow the instructions in the 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 on trigger conditions, see “Defining trigger conditions for events” on page 816.

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.

- **DatabaseStart**
  You can use the DatabaseStart event type to take action when a database is started.

- **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 [database]” on page 424.

- **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 CE.

  You may want to use disk space events to alert administrators in case 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.

  For more information, see “-fc server option” on page 143.

- **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 AUTOINCREMENT column that is near the end of its range. See "EVENT_PARAMETER function [System]" [*SQL Anywhere Server - SQL Reference*].

- **SQL errors**
  When an error is triggered, you can use the RAISERROR event type to take actions.

- **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.For more information, see "Database mirroring system events" on page 841.

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
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

One difference between event handlers and stored procedures is that event handlers do not take any arguments. Certain information about the context in which an event was triggered is available through the event_parameter function, which supplies information about the connection that caused an event to be triggered (connection ID, user ID), as well as the event name and the number of times it has been executed.

☞ For more information, 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.

☞ For more information, 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 825.

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 are not triggered solely by database actions. The free disk space types (**DBDiskSpace** and so on), as well as the **IdleTime** type, are examples.

  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

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, and consequently 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 server console.

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. Connect to the database as a user with DBA authority.
2. Select the Events folder for your database.
3. From the File menu, choose New ► Event.

   The Create Event wizard appears.
4. Follow the instructions in the wizard.

   The wizard contains many options, depending on the event you want to create. These are explained in detail 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.

   ☞ For more information, 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. Connect to the database as a user with DBA authority.
2. Select the Events folder for your database.
3. From the File menu, choose New ► Event.

   The Create Event wizard appears.

4. Type a name for the event, and then click Next.
5. Select Manually, and then click Next.
6. Select Enable This Event, and select Execute at All Databases, and then click Next.
7. Optionally, type a comment describing the event, and then click Finish to add the event to the database.
8. Enter the SQL statements for your event handler, and then choose File ► Save.

If you want to accept the default values for all remaining options, you can click Finish at an earlier stage in the wizard.

♦ 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:

- For information on triggering events, see “Triggering an event handler” on page 824.
- For information on altering events, see “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, you may have a monthly sales report scheduled, but from time to time you may want to obtain a sales report for a reason other than the end of the month.

☞ For more information on developing event handlers, see “Developing event handlers” on page 819.

♦ To trigger an event handler (Sybase Central)

1. Connect to the database as a user with DBA authority.
2. Open the Events folder for your database.

3. Select the event you want to trigger, and then choose File ► Trigger.

   The event must be enabled before you can trigger it. You can enable the event on the General tab of the Event property sheet.

   The Trigger Event dialog appears.

4. Supply any parameters the event handler requires, in a comma-separated list, as follows:

   ```
   parameter=value,parameter=value
   ```

   Click OK to trigger the event handler.

♦ 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;
   ```

   ☞ For more information, 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.

☞ For more information on developing event handlers, see "Developing event handlers" on page 819.

☞ For more information on using the debugger, see "Debugging Procedures, Functions, Triggers, and Events" [*SQL Anywhere Server - SQL Usage*].

♦ To debug an event handler

1. Start Sybase Central.

   From the Start menu, choose Programs ► SQL Anywhere 10 ► Sybase Central.

2. Connect to your database.

3. Change to Debug mode by choosing Mode ► Debug.

   The Debugger Details pane appears at the bottom of the Sybase Central main window.

4. In the Folders pane, open the Events folder.

5. Double-click the event you want to debug.

6. On the SQL tab in the right pane, click the left margin beside the line where you want to add a breakpoint. A red stop signs appears, to indicate that a breakpoint has been set. Alternatively, you can press F9 to set a breakpoint.

7. From Interactive SQL or another application, trigger the event handler using the TRIGGER EVENT statement.

8. The execution stops at the breakpoint you have set. You can now use the debugger features to trace execution, local variables, and so on.

CHAPTER 22

SQL Anywhere High Availability

Contents

About this chapter

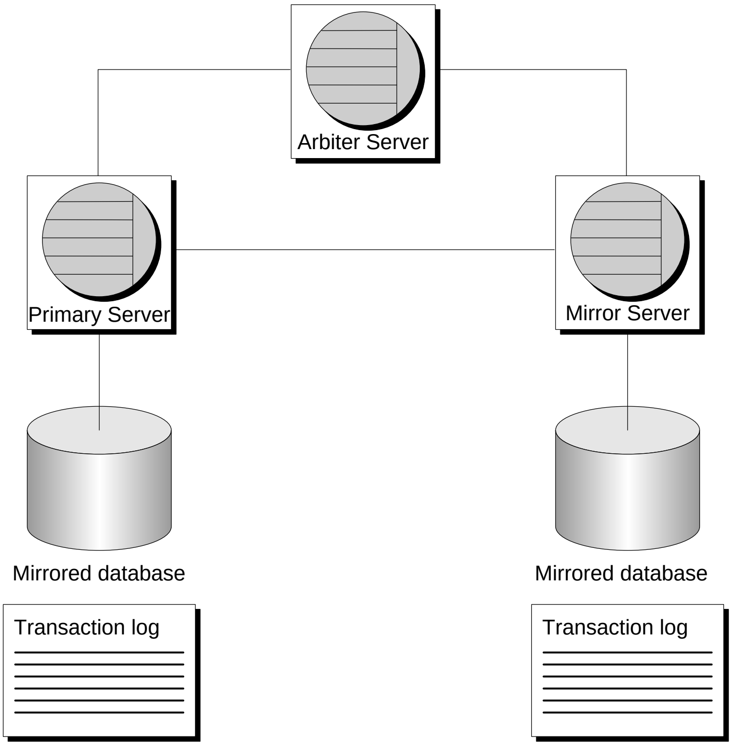
This chapter describes database mirroring and Veritas Cluster Server agents. These features provide high availability for SQL Anywhere databases.

Separately licensable option required
Use of database mirroring or a Veritas Cluster Server agent requires that you obtain the separately-licensable SQL Anywhere high availability option.
To order this component, see “Separately licensed components” [*SQL Anywhere 10 - Introduction*].

Understanding database mirroring

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 **operational servers** or **partners**.



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 cannot be accessed by clients while that server is acting as the mirror.

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 operational 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 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 console and then exits. This notifies the other servers that it has failed so that they can take appropriate action.

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 833.

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 842 and "Backup and Data Recovery" on page 761.

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 servers, you must use the network database server (dbsrv10); the personal server cannot be used.
- **LOAD TABLE statements not permitted** LOAD TABLE statements that load data into permanent tables on a mirrored database are not permitted because the data files would not be available on the mirror server. An error is reported in this case. However, LOAD TABLE is permitted on temporary tables.
- **Clients cannot connect to the database on the mirror server** If the mirror server must be stopped using the Stop Server utility (dbstop), a connection to the utility database must be used. To use the utility database, you must use the -su server option when starting the database server, or specify the utility database password in the *util_db.ini* file. See "Using the utility database" on page 272 and "Stopping a database server in a mirroring system" on page 840.
- **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, scheduled events run on the mirror server as long as failover completes before the scheduled start time for the event. Otherwise, the next scheduled occurrence of the event runs on the mirror 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 840.
- **Web servers cannot participate in a mirroring system** You cannot use a SQL Anywhere database server as a web server if the database server is participating in a database mirroring system because when failover occurs, the IP address of the database server changes.

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 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.

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 operational 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 mirroring:

- synchronous
- asynchronous
- asyncfullpage

Synchronous mode is the default. These modes control when and how transactions are recorded on the mirror server, and you set them with the -xp server option.

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.

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 acknowledgment.

Using synchronous mode provides **transaction safety** because the operational 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 acknowledgment from the mirror before replying to the application that the COMMIT has completed. Should a failure occur on the primary server, it is possible that 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 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 acknowledgment 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 autofailover option to yes using the -xp server option. Otherwise, when the failed server is restarted, it detects whether transactions were lost. If transactions were lost, it displays a message on the server console 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 840.

> **Note**
> It is recommended that you set the -xp autofailover option to yes if you are using asynchronous or asynfullpage mode. Then, if the primary server goes down, the mirror server automatically takes over as the primary server.

The synchronize_mirror_on_commit 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 acknowledgment 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 841.

See also

- "synchronize_mirror_on_commit option [database]" on page 460
- "-xp database option" on page 202
- "SET 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 an operational 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 -xf option. See "-xf server option" on page 184.

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 833.
Mode	Specifies the synchronization mode (one of synchronous, asynchronous, or page). See "Choosing a database mirroring mode" on page 831.
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 "Understanding database mirroring" on page 828.

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 will be created automatically. State information files should only be modified by the database server.

Tutorial: Using database mirroring

This tutorial shows you how to set up a database mirroring system and what happens when failover occurs. For the purposes of this tutorial, all of the database servers are running on the same computer. However in a real mirroring system, you would likely run the database servers on separate computers.

♦ To simulate failover in a database mirroring system

1. Create the following directories: *c:\server1*, *c:\server2*, and *c:\arbiter*.

2. Make a copy the sample database located in *samples-dir\demo.db*, and add it to *c:\server1*.

3. Create a transaction log for the database located in *c:\server1* by executing the following command:

   ```
   dbping -d -c UID=DBA;PWD=sql;DBF=c:\server1\demo.db
   ```

4. Make copies of the database file and transaction log in *c:\server1*, and add them to *c:\server2*.

5. Start the arbiter server by executing the following command at a command prompt:

```
dbsrv10 -x tcpip(PORT=2639) -su sql -n arbiter -xa auth=abc;DBN=demo -xf
c:\arbiter\arbiterstate.txt
```

This command line specifies the following dbsrv10 options:

- **-x** Instructs the database server to use TCP/IP communications over port 2639. The other servers also use TCP/IP, but communicate on different ports.
- **-su** Specifies the password for the utility database.
- **-n** Names the database server 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.
- **-xf** Specifies the location of the state information file for the arbiter.

6. Start server1 by executing the following command (entered all on one line) at a command prompt:

   ```
   dbsrv10 -n server1 -x tcpip(PORT=2638) -xf c:\server1\server1state.txt -
   su sql
   c:\server1\demo.db -sn mirrordemo
   -xp partner=(ENG=server2;LINKS=tcpip(PORT=2637;TIMEOUT=1));auth=abc;
   arbiter=(ENG=arbiter;LINKS=tcpip(PORT=2639;TIMEOUT=1));mode=sync
   ```

   This command line specifies the following dbsrv10 options:

   - **-n** Names the database server server1.
   - **-x** Specifies the port on which the database server runs.
   - **-xf** Specifies the location of the state information file for server1.
   - **-su** Specifies the password for the utility database.
   - **-sn** Specifies the alternate name for the database server. Both the primary and mirror server must have the same name so clients can connect without knowing in advance which server is the primary server, and which server is the mirror server.
   - **-xp** Provides information to the server that is being started so it can connect to its partner and the arbiter server.

7. Start server2 by executing the following command (entered all on one line) at a command prompt:

   ```
   dbsrv10 -n server2 -x tcpip(PORT=2637) -xf c:\server2\server2state.txt -
   su sql
   c:\server2\demo.db -sn mirrordemo
   -xp partner=(ENG=server1;LINKS=tcpip(PORT=2638;TIMEOUT=1));auth=abc;
   arbiter=(ENG=arbiter;LINKS=tcpip(PORT=2639;TIMEOUT=1));mode=sync
   ```

   This command line specifies the following dbsrv10 options:

   - **-n** Names the database server server2.
   - **-x** Specifies the port on which the database server runs.

- **-xf** Specifies the location of the state information file for server2.
- **-su** Specifies the password for the utility database.
- **-sn** Specifies the alternate name for the database server. Both the primary and mirror server must have the same name so clients can connect without knowing in advance which server is the primary server, and which server is the mirror server.
- **-xp** Provides information to the server that is being started so it can connect to its partner and the arbiter server.

8. Start Interactive SQL and connect to the primary server by running the following command:

   ```
   dbisql -c "UID=DBA;PWD=sql;ENG=mirrordemo;LINKS=tcpip"
   ```

9. 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;
   ```

10. Determine which database server you are connected to by executing the following statement:

    ```
    SELECT PROPERTY( 'ServerName' );
    ```

    The name of the primary server appears.

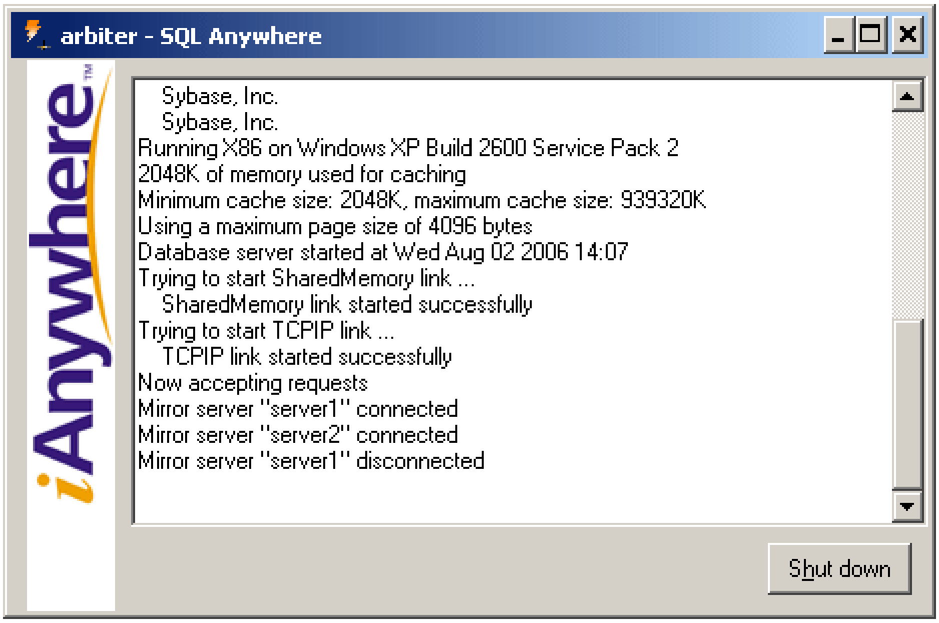
11. Initiate failover. You can do this by stopping the primary server identified in Step 10 in one of the following ways:

    - Click Shut Down in the Server Messages window.
    - Use the Windows Task Manager to end its task.
    - Issue the following command:

      ```
      dbstop -y -c "UID=DBA;PWD=sql;ENG=mirrordemo"
      ```

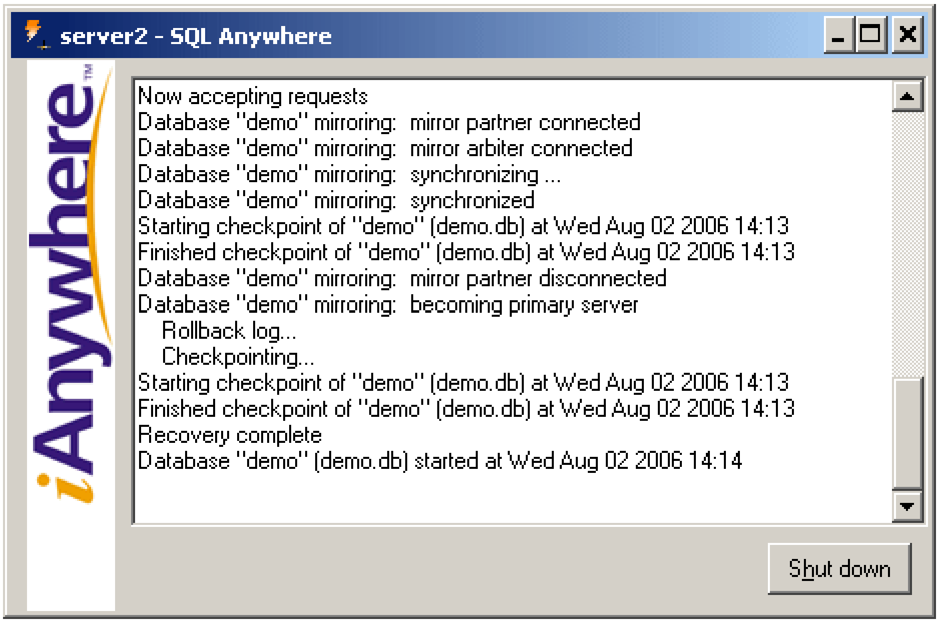
    If a warning message appears indicating that the database server still has one connection, click Yes to shut it down.

    The arbiter Server Messages window displays a message indicating that the primary server is disconnected.



```
    Sybase, Inc.
    Sybase, Inc.
Running X86 on Windows XP Build 2600 Service Pack 2
2048K of memory used for caching
Minimum cache size: 2048K, maximum cache size: 939320K
Using a maximum page size of 4096 bytes
Database server started at Wed Aug 02 2006 14:07
Trying to start SharedMemory link ...
    SharedMemory link started successfully
Trying to start TCPIP link ...
    TCPIP link started successfully
Now accepting requests
Mirror server "server1" connected
Mirror server "server2" connected
Mirror server "server1" disconnected
```

The Server Messages window for server2 displays a message indicating that it is the new primary server:



```
Now accepting requests
Database "demo" mirroring:  mirror partner connected
Database "demo" mirroring:  mirror arbiter connected
Database "demo" mirroring:  synchronizing ...
Database "demo" mirroring:  synchronized
Starting checkpoint of "demo" (demo.db) at Wed Aug 02 2006 14:13
Finished checkpoint of "demo" (demo.db) at Wed Aug 02 2006 14:13
Database "demo" mirroring:  mirror partner disconnected
Database "demo" mirroring:  becoming primary server
    Rollback log...
    Checkpointing...
Starting checkpoint of "demo" (demo.db) at Wed Aug 02 2006 14:13
Finished checkpoint of "demo" (demo.db) at Wed Aug 02 2006 14:13
Recovery complete
Database "demo" (demo.db) started at Wed Aug 02 2006 14:14
```

12. Close Interactive SQL. Click OK if you receive an error message.

13. Restart Interactive SQL by running the following command:

```
dbisql -c "UID=DBA;PWD=sql;ENG=mirrordemo;LINKS=tcpip"
```

14. Execute the following statement to see that you are now connected to the mirror server:

    ```
    SELECT PROPERTY ( 'ServerName' );
    ```

15. Execute the following statement to verify that all transactions were mirrored to the mirror database:

    ```
    SELECT * FROM test;
    ```

16. Disconnect from Interactive SQL, and then click Shut Down on the Server Messages window for the arbiter, server1, and server2 database servers.

Setting up database mirroring

The following steps assume that one database server is already running the database for which you want to set up a mirroring system.

When starting database servers that will be participating in a mirroring system, it is recommended that you include the -su option to specify the password for the utility database. Then, you can use the utility database to shut down the server, or force the mirror server to become the primary server should such a need arise. See “-su server option” on page 174.

♦ To set up a mirroring system

1. Make a copy of the database and current transaction log on a second server.

   If the existing database server is stopped, 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 591.

2. Stop the running database server and modify its command line configuration to include the mirroring options and then start the server.

   For example:

   ```
   dbsrv10 -n server1 -x tcpip(PORT=2638) -xf c:\server1\server1state.txt -
   su sql
   c:\server1\mirrordemo.db -sn mirrordemo_test
   -xp partner=(ENG=server2;LINKS=tcpip(PORT=2637;TIMEOUT=1));auth=abc;
   arbiter=(ENG=arbiter;LINKS=tcpip
   (PORT=2639;TIMEOUT=1));mode=page;autofailover=YES
   ```

3. Start another operational server.

   For example:

   ```
   dbsrv10 -n server2 -x tcpip(port=2637) -xf c:\server2\server1state.txt -
   su sql
   c:\server2\mirrordemo.db -sn mirrordemo_test
   -xp partner=(ENG=server1;LINKS=tcpip(PORT=2638;TIMEOUT=1));auth=abc;
   arbiter=(ENG=arbiter;LINKS=tcpip
   (PORT=2639;TIMEOUT=1));mode=page;autofailover=YES
   ```

4. Start the arbiter server.

   For example:

```
dbsrv10 -x tcpip -n arbiter
-xa AUTH=abc;DBN=mirrordemo -xf arbiterstate.txt
-su sql
```

Clients can now connect to the mirrored database.

Connecting to a mirrored database server

When connecting to a mirrored database, clients must use the server name that was specified by the -sn option in the commands used to start the primary and mirror servers. Using the example above (database servers were started with the option -sn mirrordemo), clients specify the connection parameter ENG=mirrordemo in their connection string:

```
…UID=user12;PWD=x92H4pY;ENG=mirrordemo;LINKS=tcpip…
```

If the primary and mirror servers are running on different subnets, then you must specify a range of IP addresses that the client should use to connect to the primary server. For example:

```
…UID=user12;PWD=x92H4pY;ENG=mirrordemo;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 connection parameter [RetryConnTO]" on page 237.

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 `LINKS=tcpip (HOST=MirrorServ1,MirrorServ2)` in the client connection string.
2. Register the servers with LDAP. See "Connecting using an LDAP server" on page 108.
3. Use the SQL Anywhere Broadcast Repeater utility (dbns10) to locate the servers. This utility listens for broadcasts and responses on one subnet, and then re-broadcasts them on another subnet. See "SQL Anywhere Broadcast Repeater utility (dbns10)" on page 665.

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 using the ALTER DATABASE statement. You must be able to connect to the utility database on the database server to use this feature. You can connect to the utility database by specifying -su option in the command to start the mirroring servers, or by specifying a password in the *util_db.ini* file. The following command forces the mirror server for the database *mymirroreddb.db* to become the primary server:

```
ALTER DATABASE mymirroreddb FORCE START;
```

☞ For more information, see "ALTER DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

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. Note that 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 272.

♦ To stop a primary, mirror, or arbiter server

- Issue a dbstop command to stop the database server.

  For example, the following command stops a server named myarbiter:

  ```
  dbstop -c "UID=DBA;PWD=sql;DBN=utility_db;LINKS=tcpip" myarbiter
  ```

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 of 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 displays message on the server console 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.

Database mirroring and transaction log files

When an operational 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 as to 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 some other means (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 643.

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='Database failover occurred',
    "message"='The following server is unavailable in the mirroring system: '
                || EVENT_PARAMETER( 'MirrorServerName' ) );
    CALL xp_stopmail ( );
END;
```

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 operational 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 back up 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*].

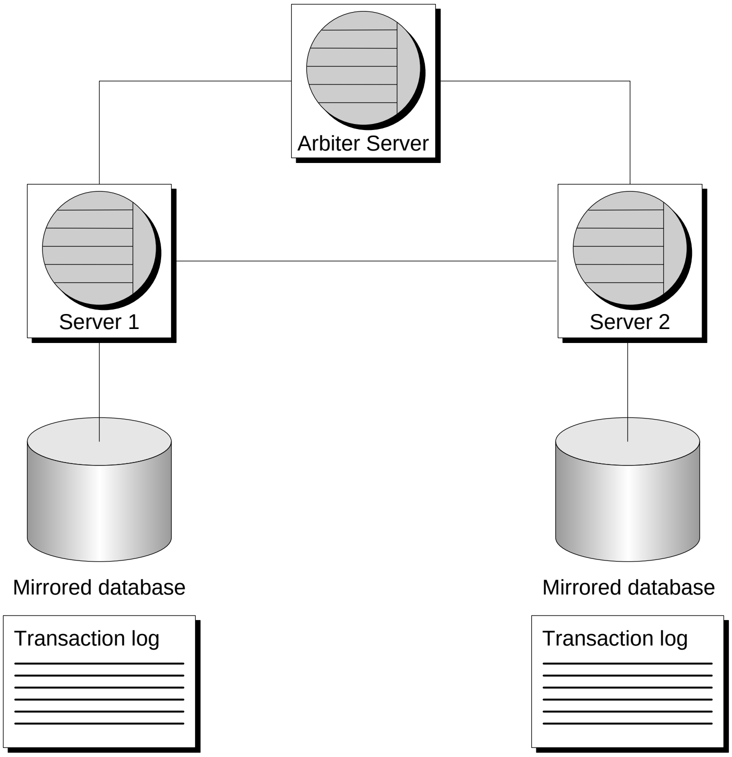
Alternatively, you can perform client-side backups using the dbbackup utility. See "Backup utility (dbbackup)" on page 591.

See also

- "Backup and Data Recovery" on page 761
- "Ensuring your database is valid" on page 776

Database mirroring scenarios

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-level properties” on page 506.

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 840.

Scenario 2: Primary server becomes unavailable and then restarts

1. The arbiter and mirror server (Server 2) detect that the primary server (Server 1) is no longer available.
2. The arbiter and Server 2 reach quorum, and Server 2 becomes the primary server.
3. Server 2 begins accepting client connections.

4. Server 1 comes back online and reconnects to Server 2 and the arbiter.
5. Server 1 requests quorum, but Server 2 is already the primary server.
6. Server 1 is the mirror server, and waits for changes from Server 2.
7. 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 840.

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.

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

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.ianywhere.com/developer/technotes/asa_cluster_db_service.html.

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 10.0.0 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 or by adding the following line to the *install-dir\win32\util_db.ini* file on each node of the cluster:

  ```
  pwd = [password-of-your-choice]
  ```

  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 will 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*
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.

   The Add Resource dialog appears.

b. Choose SAServer from the Resource Type dropdown list.

   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. Type a name in the Resource Name field.

d. Add the following attribute values to the following attributes:

   ♦ **cmdStart** dbsrv10 -x tcpip *database-file-on-shared-disk* -n *server-name*

   ♦ **cmdMonitor** dbping -c "ENG=*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 machine 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;ENG=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 to 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.
8. Add the SADatabase agent:

   a. Choose File ► Import Types.

   b. Navigate to *%VCS_HOME%\conf\config\*, and then 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.

   The Add Resource dialog appears.

   b. Choose SADatabase from the Resource Type dropdown list.

   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. Type a name in the Resource Name field.

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 machine 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;ENG=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;ENG=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;ENG=VCS2;DBN=DEMO;LINKS=tcpip"
```

4. Reconnect to Interactive SQL. You should be able to connect and execute the query successfully.

Part VI. Security

This section describes security features in SQL Anywhere

CHAPTER 23

Keeping Your Data Secure

Contents

About this chapter

This chapter describes SQL Anywhere features that help make your database secure. It also presents overviews of other security features, providing pointers to where you can find more detailed information.

☞ For information about encrypting client/server communications, see "Transport-Layer Security" on page 885.

Security features overview

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 340.
- **Discretionary access control features** These features control the actions a user can perform while connected to a database. See "Database permissions overview" on page 336.
- **Auditing** This feature helps you maintain a record of actions on the database. See "Auditing database activity" on page 868.
- **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 21.
- **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 361.
- **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 database option" on page 197 and "DatabaseKey connection parameter [DBKEY]" on page 217.

  Table encryption features allow you to encrypt individual tables, instead of encrypting the entire database. See "Table encryption" on page 879.
- **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 885.

  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-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.
  See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].
- **Secured features** You can disable features for all databases running on a database server.

Database administrators are responsible for data security. In this chapter, unless otherwise noted, you require DBA authority to perform the tasks described.

☞ User IDs and permissions are security-related topics. For information, see “Managing User IDs and Permissions” on page 335.

Security tips

As database administrator, there are many actions you can take to improve the security of your data. For example, you can:

- **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.

- **Require long passwords** You can set the min_password_length public option to disallow short (and therefore easily guessed) passwords.

  For information, see "min_password_length option [database]" on page 432.

- **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 DBA 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 server option" on page 170 and "Specifying secured features" on page 866.

- **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 authorities other than those they have been given by the operating system. In a security-conscious environment, you should drop these functions.

  For information on dropping procedures, see "DROP 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.

  For more information, see "Running the server outside the current session" on page 33.

- **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 connections to use TCP/IP, which is more secure than the shared memory connection.

- **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 server option" on page 142 and "-ek database option" on page 197.

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 the following:

- "Managing User IDs and Permissions" on page 335
- "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 the following:

- "Security concerns: Unrestricted database access" on page 91
- "Using integrated logins" on page 84
- "login_mode option [database]" on page 422

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.

☞ For information about using the verify_password_function option, see "verify_password_function option [database]" on page 470.

Implement minimum password lengths

By default, passwords can be any length. For greater security, you can enforce a minimum length requirement on all new 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;
```

☞ For more information, see "min_password_length option [database]" on page 432.

Implement password expiration

By default, database passwords never expire. You can use the post_login_procedure option to implement such functionality as warning users that their password is going to expire or changing passwords. See "post_login_procedure [database]" on page 443.

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.

☞ For information on creating ODBC data sources, see "Creating an ODBC data source" on page 65.

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 608.

Use a password verification function

You can use the verify_password_function option to specify a function that implements password rules. See "verify_password_function option [database]" on page 470.

The following example defines a number of procedures and functions. Together they implement advanced password rules that include requiring certain types of characters in the password, disallowing password reuse, and expiring passwords. These procedures and functions are called by the database server with the verify_password_function and login_procedure options when a user ID is created or a password is changed, and when a user connects. 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*.

```
-- only DBA should have permissions on this table
CREATE TABLE DBA.t_pwd_history(
        pk          INT         DEFAULT AUTOINCREMENT PRIMARY KEY,
        change_date TIMESTAMP   DEFAULT CURRENT TIMESTAMP,  -- when pwd set
        grace_date  DATE,               -- a day after password expires to
                                        -- allow user to log in
        user_name   CHAR(128),          -- the user whose password is set
        pwd_hash    CHAR(32) );         -- hash of password value to detect
                                        -- duplicate passwords

-- called on every GRANT CONNECT TO ... IDENTIFIED BY ... statement
-- 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;

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';

-- called on every connection to check for password expiry
CREATE PROCEDURE DBA.p_login_check()
BEGIN
    DECLARE uid                     CHAR(128);
    DECLARE INVALID_LOGON           EXCEPTION FOR SQLSTATE '28000';
    DECLARE last_pwd_change         DATE;
    DECLARE grace_date              DATE;
    DECLARE is_dba                  CHAR;
    DECLARE msg_str                 CHAR(255);

    SET uid = CONNECTION_PROPERTY( 'Userid' );
    IF ( EXISTS( SELECT * FROM t_pwd_history WHERE user_name = uid ) ) THEN
        SELECT FIRST  t.change_date, t.grace_date
                INTO last_pwd_change, grace_date
                FROM t_pwd_history t WHERE t.user_name = uid
                ORDER BY t.change_date DESC;
    END IF;
    IF last_pwd_change IS NULL THEN
        -- no password change in t_pwd_history, so create one now.
        INSERT INTO t_pwd_history( user_name, pwd_hash )
            VALUES( uid, 'unknown' );
        COMMIT WORK;
    ELSE
        IF EXISTS( SELECT * FROM SYS.SYSUSERAUTHORITY a, SYS.SYSUSER u
                   WHERE u.user_name = uid AND u.user_id = a.user_id AND
                         a.auth = 'DBA' ) THEN
            SET is_dba = 'Y';
        ELSE
            SET is_dba = 'N';
        END IF;

        -- remove any locks on t_pwd_history and SYSUSERAUTHORITY
        ROLLBACK WORK;
        -- check if last password change was over five months ago
        IF CURRENT DATE > DATEADD( month, 5, last_pwd_change ) THEN
            -- Never expire DBA accounts so that the database does not
            -- get locked out by all users.
            IF CURRENT DATE < DATEADD( month, 6, last_pwd_change ) OR
               is_dba = 'Y' OR
               ( grace_date IS NOT NULL AND grace_date = CURRENT DATE ) THEN
```

```
                SET msg_str = 'The password for user ' || uid ||
                              ' expires on ' ||
                              CAST( DATEADD( month, 6, last_pwd_change )
                                    AS DATE ) ||
                              '.  Please change it before it expires.';
                MESSAGE msg_str;
                -- The post_login_procedure option is set to
                -- p_post_login_check, which will cause a dialog to be
                -- displayed notifying the user their password will
                -- expire soon. dbisql and dbisqlc will display this
                -- dialog, and user applications can call the
                -- post_login_procedure and display this dialog.
                -- May want to use xp_send_mail to notify user and/or
                -- administrator.
            ELSEIF grace_date IS NULL THEN
                -- Allow one grace login day. The first login on the grace
                -- day fails to ensure the user knows their password has
                -- expired
                UPDATE t_pwd_history t SET t.grace_date = CURRENT DATE
                    WHERE t.grace_date IS NULL AND t.user_name = uid;
                COMMIT WORK;
                SET msg_str = 'The password for user ' || uid ||
                              ' has expired, but future logins will ' ||
                              'be allowed today only so that the password '
||

                              'can be changed.';
                MESSAGE msg_str;
                RAISERROR 28000 msg_str;
                RETURN;
            ELSE
                SET msg_str = 'The password for user ' || uid ||
                              ' has expired and must be reset by your DBA.';
                MESSAGE msg_str;
                -- may want to use xp_send_mail to notify administrator.
                RAISERROR 28000 msg_str;
                RETURN;
            END IF;
        END IF;
    END IF;

    CALL sp_login_environment;
END;

GRANT EXECUTE ON DBA.p_login_check TO PUBLIC;
SET OPTION PUBLIC.login_procedure = 'DBA.p_login_check';

-- called by dbisql, dbisqlc and some user applications on every successful
-- connection to check for warnings which should be displayed
CREATE PROCEDURE DBA.p_post_login_check()
RESULT( warning_text VARCHAR(255), warning_action INT )
BEGIN
    DECLARE uid             CHAR(128);
    DECLARE last_pwd_change DATE;
    DECLARE warning_text    CHAR(255);
    DECLARE warning_action  INT;

    SET uid = CONNECTION_PROPERTY( 'Userid' );
    SELECT FIRST t.change_date
            INTO last_pwd_change
            FROM t_pwd_history t WHERE t.user_name = uid
            ORDER BY t.change_date DESC;
    IF CURRENT DATE > DATEADD( month, 5, last_pwd_change ) THEN
        SET warning_text = 'Your password expires on ' ||
                           CAST( DATEADD( month, 6, last_pwd_change )
                                 AS DATE ) ||
                           '.  Please change it before it expires.';
        SET warning_action = 1;
    ELSE
        -- There is no warning
        SET warning_text = NULL;
        SET warning_action = 0;
    END IF;
    -- Return the warning (if any) through this result set
    SELECT warning_text, warning_action;
END;

GRANT EXECUTE ON DBA.p_post_login_check TO PUBLIC;
SET OPTION PUBLIC.post_login_procedure = 'DBA.p_post_login_check';
```

Controlling the tasks users can perform

Users can access only those objects to which they have been granted access.

You grant permission on an object to another user with the GRANT statement. You can also delegate permission granting privileges on an object to other users.

The GRANT statement also gives more general permissions to users:

- Granting CONNECT permissions to a user allows them to connect to the database.
- Granting RESOURCE authority allows the user to create tables, views, procedures, and so on.
- Granting DBA authority to a user gives that user the ability to see and do anything in the database. The DBA also uses the GRANT statement to create and administer groups.

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.

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.

☞ For more information, see:

- “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.

☞ For more information on these features, see:

- "Benefits of procedures and triggers" [*SQL Anywhere Server - SQL Usage*]
- "Using views and procedures for extra security" on page 361

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 server option" on page 170.

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.

   ```
   dbsrv10 -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;ENG=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:

```
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 server option" on page 170
- "-sk server option" on page 173
- "secure_feature_key [database]" on page 455
- "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.

☞ For information about the transaction log, see "The transaction log" on page 766.

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.

Turning on auditing

The database administrator can turn on auditing to add security-related information to the transaction log.

Auditing is off by default. To enable auditing on a database, the DBA must set the value of the auditing public option to On. Auditing then remains enabled until explicitly disabled, by setting the value of the auditing option to OFF. You must have DBA permissions to set this option.

♦ To turn on auditing

1. Connect to your database as the DBA.
2. Execute the following statement:

```
SET OPTION PUBLIC.auditing = 'On';
```

☞ For more information, see "auditing option [database]" on page 395.

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 [database]” on page 424 and “conn_auditing option [database]” on page 403.

Retrieving audit information

You can use the Log Translation (dbtran) utility to retrieve audit information. You can access this 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 of the transactions, along with some information on what user executed each command. By using the -g option, dbtran includes more comments containing the auditing information. The -g option automatically sets 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.

You can run the Log Translation utility against a running database server or against a database log file.

♦ To retrieve auditing information from a running database server

1. Make sure your user ID has DBA authority.
2. With the database server running, execute the following statement at a system command prompt:

   ```
   dbtran -g -c "UID=DBA;PWD=sql;..." -n demo.sql
   ```

   ☞ For information about connection strings, see “Connection parameters” on page 206.

♦ To retrieve auditing information from a transaction log file

1. Close the database server to ensure the log file is available.
2. At a system command prompt, execute the following statement to place the information from the file *demo.log* and into the file *demo.sql*.

   ```
   dbtran -g demo.log
   ```

☞ For more information, see “The Log Translation utility” on page 637.

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 permissions 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.

An auditing example

This example shows how the auditing feature records attempts to access unauthorized information.

1. As database administrator, turn on auditing.

   You can do this from Sybase Central as follows:

   - Connect using the SQL Anywhere 10 Demo data source. This connects you as the DBA user.
   - From the Context dropdown list, choose **demo - DBA** to ensure the SQL Anywhere sample database is selected, and then from the File menu, choose Options.
   - Select auditing from the list of options, and type the value **On** in the Value field. Click Set Permanent Now to set the option and then click Close.

   Alternatively, you can use Interactive SQL. Connect to the sample database from Interactive SQL as the DBA user and execute the following statement:

   ```
   SET OPTION PUBLIC.auditing = 'On';
   ```

2. Add a user to the sample database, named BadUser, with password **BadUser**. You can do this from Sybase Central. Alternatively, you can use Interactive SQL and enter the following statement:

   ```
   GRANT CONNECT TO BadUser
   IDENTIFIED BY 'BadUser';
   ```

3. Use Interactive SQL to connect to the sample database as BadUser and attempt to access confidential information in the **Employees** table with the following query:

   ```
   SELECT Surname, Salary
   FROM GROUPO.Employees;
   ```

   You receive an error message: `Permission denied: you do not have permission to select from "Employees"`.

4. At a command prompt, execute the following command:

   ```
   dbtran -g -c "DSN=SQL Anywhere 10 Demo" -n demo.sql
   ```

   This command produces a file named *demo.sql*, which contains the transaction log information and a set of comments holding audit information. The lines that indicate the unauthorized BadUser attempt to access the Employees table are included in the file as follows:

```
--AUDIT-1001-0000287812 -- 2004/02/11 13:59:58.765 Checking Select
permission on Employees - Failed
--AUDIT-1001-0000287847 -- 2004/02/11 13:59:58.765 Checking Select
permission on Employees(Salary) - Failed
```

5. Restore the sample database to its original state so other examples you try in this documentation give the expected results.

   Connect as the DBA user, and perform the following operations:

   - Revoke Connect privileges from the user ID **BadUser**.
   - Set the PUBLIC.auditing option to 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

- “The Log Translation utility” on page 637
- “The Transaction Log utility” on page 685

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 permissible values are DBA, all, or none. See "-gd server option" on page 146.

- **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 server option" on page 154.

- **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 Shutdown on the 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 server option" on page 148.

- **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 server option" on page 148.

- **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 885.

- **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 server option" on page 170.

Encrypting 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.

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 -e option.

  The following example creates the database *test.db* using simple encryption:

  ```
  dbinit -e test.db
  ```

☞ For more information, see "Initialization utility (dbinit)" on page 614.

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
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.

The encryption algorithm

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). It has many properties that lend itself well to encryption of SQL Anywhere databases in terms of performance and size.

You can also specify a separate FIPS-approved AES algorithm for strong encryption using the AES_FIPS type. When the database server is started with the -fips option, you can run databases encrypted with AES or AES_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 server option" on page 144.

Note
FIPS is not available on all platforms. For a list of supported platforms, see the Separately licensed components table in SQL Anywhere 10.0.0 Components by Platform

The SQL Anywhere security option must be installed on any computer used to run a database encrypted with AES_FIPS.

Separately licensed component required
ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.
See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].

Creating a strongly-encrypted database

To create a new database with strong encryption, you can use:

- The Database Initialization utility (dbinit) in combination with various options to enable strong encryption.

  The dbinit utility -ek option and -ep 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 AES_FIPS for the FIPS-approved algorithm.

  For more information, see "Initialization utility options" on page 616 and "The Initialization utility" on page 614.

- The ENCRYPTION clause in the CREATE DATABASE statement. The KEY option sets the encryption key and the ALGORITHM option sets the encryption algorithm to AES or AES_FIPS for the FIPS-approved algorithm.

  You can also use the Sybase Central Create Database wizard to create a strongly encrypted database.

  For more information, see "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

- 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 prompt box or on the command line use the -ek or -ep option. The -ea option sets the encryption algorithm to AES or AES_FIPS for the FIPS-approved algorithm.

  You can also use the Sybase Central Unload Database wizard to create a strongly encrypted database.

  For more information, see "Unload utility options" on page 694 and "The Unload utility" on page 689.

♦ To create a strongly 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 AES encryption.

   ```
   CREATE DATABASE 'c:\\myencrypteddb'
   TRANSACTION LOG ON
   ENCRYPTED ON
     KEY '0kZ2o52AK#'
     ALGORITHM 'AES_FIPS';
   ```

♦ To create a strongly encrypted database (command prompt)

1. At a command prompt, 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 dialog box, 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. Start the database from the command prompt.

   ```
   dbeng10 myencrypteddb.db -ek "0kZ2o56AK#"
   ```

☞ For more information about the encryption key, see "DatabaseKey connection parameter [DBKEY]" on page 217.

If you have a database you want to encrypt, you can do so using the CREATE ENCRYPTED FILE statement. You are not actually overwriting the file, you are creating a copy of the file in encrypted form.

Note
You cannot encrypt a database if table encryption is enabled. Instead, you must recreate the database without table encryption.

♦ To encrypt a database after it has been created

1. Encrypt an unencrypted database using the CREATE ENCRYPTED FILE statement.

   The following example takes the database file *current.db*, and creates an encrypted copy of it named *encrypted.db*.

   ```
   CREATE ENCRYPTED FILE encrypted.db
   FROM current.db
   KEY abc
   ALGORITHM AES;
   ```

2. Using the same encryption key information, and following the file name convention you used for the database file, encrypt the associated transaction log file(s), dbspace file(s), and mirror log file (if any), using the CREATE ENCRYPTED FILE statement. See "CREATE ENCRYPTED FILE statement" [*SQL Anywhere Server - SQL Reference*].

Note
Although you can use the CREATE ENCRYPTED FILE statement to encrypt an unencrypted database, you cannot use the statement to enable only table encryption for a database that does not have encryption enabled. To enable encryption on a database, you must recreate the database and enable table encryption. See "Enabling table encryption" on page 880.

You can decrypt a database using the CREATE DECRYPTED FILE statement. As with the CREATE ENCRYPTED FILE statement, you are creating a copy of the file (in this case, in decrypted form), and not actually overwriting the file. You must remember to decrypt not only the database file, but also the associated transaction log files, and dbspace(s). See "CREATE DECRYPTED FILE statement" [*SQL Anywhere Server - SQL Reference*].

Working with encryption keys

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 includes between 8 and 30 characters, a combination of upper and lowercase characters, and numbers, letters, and special characters.

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.

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 FILE 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

1. Change the encryption key for an encrypted database using the CREATE ENCRYPTED FILE statement.

   The following example takes the database file *currentkey.db*, encrypted with key abc, and creates a copy of it called *newkey.db*, encrypting it with the key abc123.

   ```
   CREATE ENCRYPTED FILE newkey.db
   FROM currentkey.db
   KEY abc123
   OLD KEY abc
   ALGORITHM AES;
   ```

2. Using the same encryption key information, and following the file name convention you used for the database file, encrypt the associated transaction log file(s), dbspace file(s), and mirror log file (if any), using the CREATE ENCRYPTED FILE statement. See "CREATE ENCRYPTED FILE statement" [*SQL Anywhere Server - SQL Reference*].

Choosing the encryption key

As with most passwords, it is best to choose a 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.

You must supply this key each time you want to start the database. Lost or forgotten keys result in completely inaccessible databases.

Protecting the encryption key

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. In cases where the database administrator starts the database, clients never need to have access to the key.

☞ For more information, see "-ep server option" on page 142.

Controlling strong encryption

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.

Strong encryption status

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 FILE statement on an existing database.

You can rebuild the database to unload all of 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

- "Reloading databases" [*SQL Anywhere Server - SQL Usage*]
- "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]

Performance issues

Performance of SQL Anywhere is somewhat 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.

See also

- "Use the cache to improve performance" [*SQL Anywhere Server - SQL Usage*]
- "-c server option" on page 127

Encrypting portions of a database

If you want to encrypt only portions of 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 will require this key each time you want to decrypt the data.

Caution
Protect your key. Be sure to store a copy of your key in a safe location. A lost key will result in the encrypted data becoming completely inaccessible, from which there is no recovery.

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, as well as 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';
```

☞ For more information about the ENCRYPT and DECRYPT functions, see “Alphabetical list of functions” [*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.

For information about encrypting materialized views, see “Encrypting and decrypting 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 a database” on page 873.

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 temporary files for the database.

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 614.

Enabling table encryption

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 simple table encryption (SQL)

- Create a database using the CREATE DATABASE statement, but do not include key or algorithm settings.

  The following command creates the database *new.db* with simple encryption enabled for tables:

  ```
  CREATE DATABASE new.db ENCRYPTED TABLE;
  ```

  Later, when you encrypt a table in this database, the simple encryption algorithm is used.

♦ To create a database with strong 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 AES_FIPS encryption algorithm:

  ```
  CREATE DATABASE new.db
  ENCRYPTED TABLE
  KEY abc
  ALGORITHM AES_FIPS;
  ```

  Later, when you encrypt a table in this database, the AES_FIPS algorithm is used, as well as the key abc.

♦ To create a database with simple table encryption (command prompt)

- Create a database with the dbinit -et option, but do not include a key or encryption algorithm.

  The following command creates the database *new.db* with simple encryption enabled for tables:

  ```
  dbinit new.db -et
  ```

  Later, when you encrypt a table in this database, the simple encryption algorithm is used.

♦ To create a database with strong table encryption (command prompt)

- Create a database with the dbinit -et and -ek options, and specifying a key and an encryption algorithm.

  The following command creates the database *new.db* with strong encryption enabled for tables that use the key abc, and the AES_FIPS encryption algorithm:

  ```
  dbinit new.db -et -ek abc -ea AES_FIPS
  ```

  Later, when you encrypt a table in this database, the AES_FIPS algorithm is used, as well as the key abc.

See also

- ♦ "Initialization utility (dbinit)" on page 614
- ♦ "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*]
- ♦ "Create Database wizard" on page 614

Encrypting tables

To encrypt tables in your database, table encryption must already be enabled in the database.

When you encrypt a table, the table encryption settings that were specified at database creation time, such as simple, AES, FIPS, and so on, are applied to the table.

♦ To encrypt a table at table creation

- Create a table using the ENCRYPTED clause of the CREATE TABLE statement.

The following command creates a encrypted table Employees:

```
CREATE TABLE Employees (
 MemberID CHAR(40),
 CardNumber INTEGER )
ENCRYPTED;
```

♦ To decrypt a table after it has been created

- Decrypt a table with the NOT ENCRYPTED clause of the ALTER TABLE statement.

  The following command decrypts the Employees table:

  ```
  ALTER TABLE Employees
  NOT ENCRYPTED;
  ```

See also

- “Encrypting a database” on page 873
- “Enabling table encryption” on page 880
- “CREATE TABLE statement” [*SQL Anywhere Server - SQL Reference*]
- “ALTER TABLE statement” [*SQL Anywhere Server - SQL Reference*]

Keeping your Windows CE database secure

This section describes SQL Anywhere features that help make your Windows CE 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 detailed information.

Many of the SQL Anywhere security features for Windows desktop platforms are supported on Windows CE, such as database file encryption and simple communication encryption, or have modified support, such as the Log Translation utility.

Databases running on Windows CE uses 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.

☞ For more information, see “Controlling database access” on page 860.

Windows CE device security

If you are storing sensitive data on your Windows CE device, you may want to use the security features provided for your Windows CE device.

☞ For more information on available security features, see the User's Manual provided with your Windows CE 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 dialog when you start the database on your Windows CE device.

☞ For more information, see “Controlling permissions from the command line” on page 21.

☞ For information on setting options on Windows CE, see “Specifying server options on Windows CE” on page 985.

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 CE, so you cannot translate a log stored on a Windows CE device. Copy the transaction log file to your PC to use this utility.

☞ For more information, see “Auditing database activity” on page 868.

Database encryption on Windows CE

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 CE.

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 scrambles 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. If you are encrypting a database to use on Windows CE, it must be encrypted with the AES algorithm.

☞ For more information, see "Encrypting a database" on page 873.

Communication encryption and Windows CE

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.

Simple communication encryption accepts communication packets that are encrypted with simple encryption. This level of communication encryption is supported on all platforms, including Windows CE and on previous versions of SQL Anywhere.

Strong communication encryption is not available on Windows CE.

☞ For more information about encrypting communications, see "Encryption connection parameter [ENC]" on page 222.

CHAPTER 24

Transport-Layer Security

Contents

About this chapter

This chapter shows you how to 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.

For information about database file encryption, see:

- SQL Anywhere databases: "Encrypting a database" on page 873
- UltraLite databases: "Security considerations" [*UltraLite - Database Management and Reference*]

Separately licensed component required
ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.
See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].

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.

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 public certificates and identity certificates for each server. You can use server authentication for SQL Anywhere server-client communication or for MobiLink synchronization:

  - For SQL Anywhere server-client 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.

Separately licensed component required
ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.
See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].

TLS support

RSA, ECC, and FIPS encryption are not available on all platforms.

RSA encryption

RSA encryption is provided free with SQL Anywhere and can be used for client/server communication, synchronization, and web services. RSA can be used alone or with FIPS, but FIPS requires a separate license.

For a list of supported platforms for RSA, see:

- RSA for SQL Anywhere client/server communication: SQL Anywhere table in SQL Anywhere 10.0.0 Components by Platform.
- RSA for UltraLite MobiLink clients: UltraLite table in SQL Anywhere 10.0.0 Components by Platform.
- RSA for MobiLink synchronization: MobiLink table in SQL Anywhere 10.0.0 Components by Platform.

ECC encryption

☞ For a list of supported platforms for ECC, see the Separately licensed components table in SQL Anywhere 10.0.0 Components by Platform.

Separately licensed component required
ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.
See “Separately licensed components” [*SQL Anywhere 10 - Introduction*].

FIPS-approved encryption technology

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 Communication Security Establishment (CSE).

☞ FIPS technology requires a separate license. See “Separately licensed components” [*SQL Anywhere 10 - Introduction*].

SQL Anywhere uses two FIPS-certified modules for encryption, both from Certicom. On Windows CE and Palm OS, SQL Anywhere uses Certicom Security Builder Government Services Edition v1.0.1. This is number 316 on the page http://csrc.nist.gov/cryptval/140-1/140val-all.htm. On Windows desktop and Unix, SQL Anywhere uses Certicom Security Builder FIPS Module v2.0. This is number 542 on the same page.

For transport-layer security, FIPS only works with RSA encryption. (ECC is not yet covered by the FIPS program.)

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 or an error is issued. 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 a list of supported platforms for FIPS, see the Separately licensed components table in SQL Anywhere 10.0.0 Components by Platform.

☞ For information about encrypting SQL Anywhere database files with FIPS technology, see “Strong encryption” on page 873.

Certificates

SQL Anywhere includes a tool called gencert that allows you to create X.509 certificates 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.

Separately licensed component required

ECC encryption and FIPS-approved encryption require a separate license. All strong encryption technologies are subject to export regulations.

See “Separately licensed components” [*SQL Anywhere 10 - Introduction*].

Setting up transport-layer security

The following steps provide an overview of the tasks required to set up transport-layer security.

♦ Overview of setting up transport-layer security

1. Create digital certificates.

   You create identity and public certificates. The server identity certificate contains the server's private key and should be stored securely with the database or MobiLink server. You distribute the server's public certificate to your clients.

   ☞ See "Creating digital certificates" on page 890.

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 certificate, and the password to protect the server's private key.

     ☞ See "Starting the database server with transport-layer security" on page 898.

   - **Configure client applications to use transport-layer security** Specify the path and file name of trusted public certificates using the Encryption connection parameter [ENC].

     ☞ See "Configuring client applications to use transport-layer security" on page 899.

3. If you are setting up transport-layer security for MobiLink synchronization:

   - **Start the MobiLink server with transport-layer security** Use the mlsrv10 -x option to specify the security stream, the server identity certificate, and the password to protect the server's private key.

     ☞ See "Starting the MobiLink server with transport-layer security" on page 904.

   - **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 public server certificates.

     ☞ See "Configuring MobiLink clients to use transport-layer security" on page 905.

Other resources for getting started

- http://www.sybase.com/detail?id=1009621

Creating digital certificates

To set up transport-layer security you must generate digital certificates.

SQL Anywhere Certificate Generation utility

You can use the SQL Anywhere certificate generation utility, gencert, to generate X.509 certificates using RSA or ECC.

☞ For more information, see "Certificate generation utility [gencert]" on page 912.

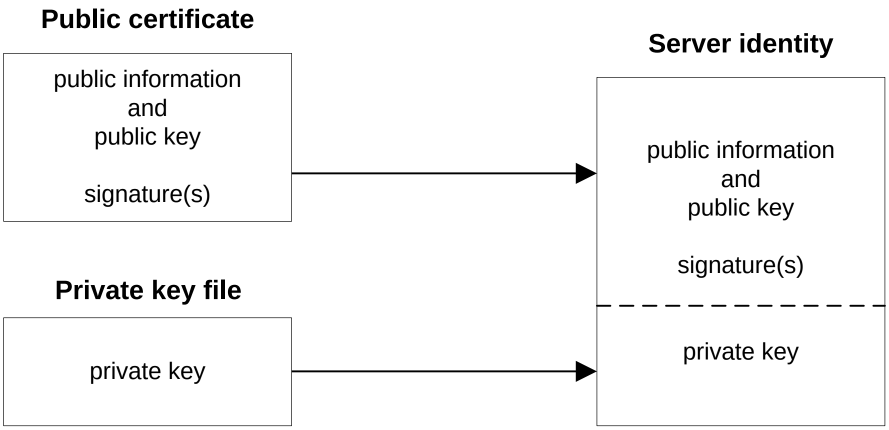
SQL Anywhere Certificate Reader utility

You can use the SQL Anywhere certificate reader utility, readcert, to read X.509 certificates using RSA or ECC.

For more information, see "Certificate reader utility [readcert]" on page 911.

Certificates for server and client authentication

You can follow the same process to create certificates for server or client authentication. In each case, you create identity certificates and public certificates.



You can create a server identity certificate by concatenating a public certificate and the matching private file.

For server authentication, you create a server identity certificate and public certificate to distribute to clients.

Certificate configurations

The public certificate can be self-signed or signed by a commercial or enterprise Certificate Authority.

- **Self-signed certificates** Self-signed server or client certificates can be used for simple setups. In this case, the private key used to sign public certificates is stored with your MobiLink or database server (for server authentication) instead of a commercial Certificate Authority or dedicated facility.

  ☞ See "Self-signed root certificates" on page 891.

- **Enterprise root certificates** Enterprise root certificates improve data integrity and extensibility for multi-server deployments.

  - You can store the private key used to sign public certificates in a secure central location.
  - For server authentication, you can add MobiLink or database servers without reconfiguring clients.

  ☞ See "Certificate chains" on page 892.

- **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 892 and "Globally-signed certificates" on page 894.

Self-signed root certificates

Self-signed root certificates can be used for simple setups involving a single MobiLink or database server. In this case, the private key used to sign trusted public certificates is stored with your MobiLink or database server instead of a dedicated facility.

> **Tip**
> Use enterprise level certificate chains or commercial certificate authorities if you require multiple identity certificates. 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 892.

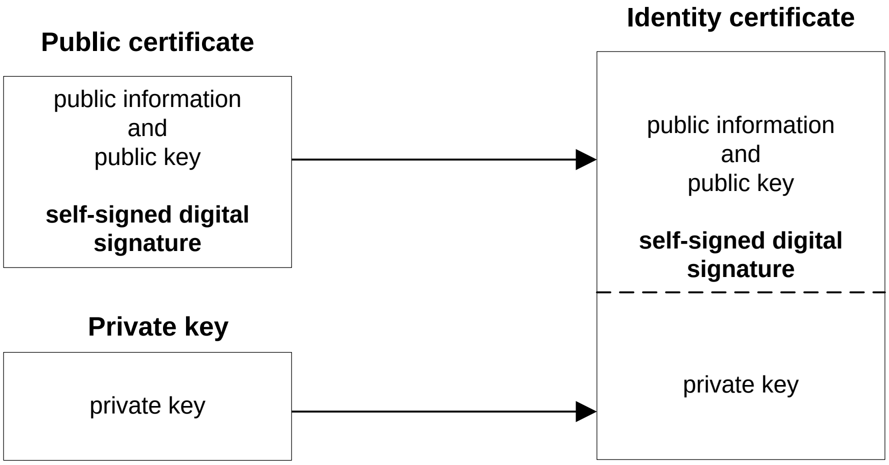
To set up self-signed certificates, you generate the following certificates using the gencert utility:

- **Public certificate** For server authentication certificates, the self-signed public 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.

  ☞ For more information about MobiLink, see digital signatures and server authentication, see "Server authentication" on page 905.

  ☞ For more information about SQL Anywhere servers, see "Starting the database server with transport-layer security" on page 898.

- **Identity certificate** For server authentication certificates, the identity certificate is stored securely with a MobiLink or database server. It is a combination of the self-signed public 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.



☞ For information about how to generate self-signed root certificates, see "Certificate generation utility [gencert]" on page 912.

Certificate chains

If you require multiple identity certificates, 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 identity certificates.

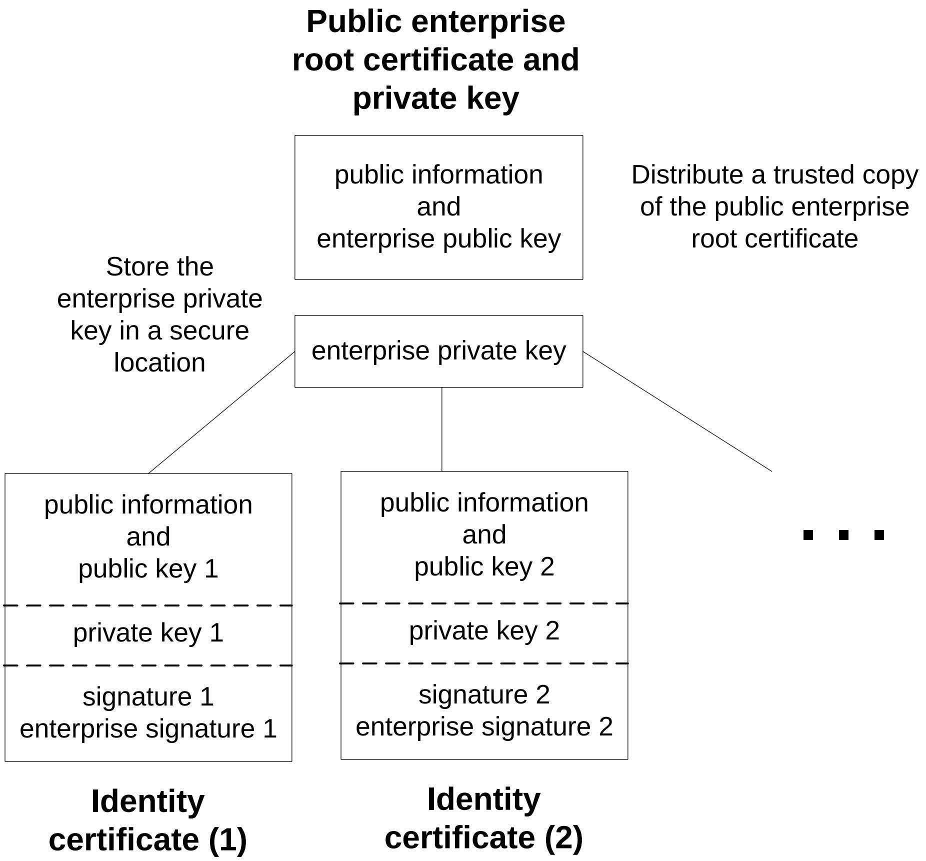
☞ For more information about self-signed certificates, see "Self-signed root certificates" on page 891.

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 public certificate.
- **Security** The enterprise root certificate's private key does not reside with identity certificates. 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.


Public enterprise root certificate and private key
public information and enterprise public key
Distribute a trusted copy of the public enterprise root certificate
Store the enterprise private key in a secure location
enterprise private key
public information and public key 1
private key 1
signature 1 enterprise signature 1
public information and public key 2
private key 2
signature 2 enterprise signature 2
. . .
Identity certificate (1)
Identity certificate (2)

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 identity certificates.

  Use the public enterprise root certificate and enterprise private key to sign each identity certificate. For server authentication, the identity certificate is used for the server.

☞ For more information about creating certificates, see "Certificate generation utility [gencert]" on page 912.

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.

☞ For more information, see "Globally-signed certificates" on page 894.

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 public 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 identity certificates.

☞ For information about creating server certificates, see “Signed identity certificates” on page 894.

☞ For information about generating enterprise root certificates, see “Globally-signed certificates” on page 894.

Signed identity certificates

You can use an enterprise root certificate to sign server identity certificates.

For server authentication, you generate identity certificates for each server. Since these certificates are signed by an enterprise root certificate, you use the gencert -s option.

☞ For information about generating signed identity certificates, see “Certificate generation utility [gencert]” on page 912.

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:

- In the case of 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 certificates, you:

- Create a certificate request using the SQL Anywhere certificate request utility, reqcert.

- Use a Certificate Authority to sign each request. You can combine the signed request with the corresponding private key to create the server identity certificate.

☞ See "Using reqtool to obtain global certificates" on page 895.

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 reqtool to obtain global certificates

SQL Anywhere transport-layer security is based on Certicom SSL/TLS Plus libraries, which require ECC or RSA certificates. You can obtain a global certificate from any Certificate Authority that can supply X. 509 certificates.

There are several ways to obtain certificates. One way is to use the reqtool utility, which is installed when you install the security component. This tool creates a server's private key and a global certificate request.

Example

The following example creates an ECC certificate request:

```
> reqtool
-- Certicom Corp. Certificate Request Tool 3.0d1 --
Choose certificate request type:
  E - Personal email certificate request.
  S - Server certificate request.
  Q - Quit.
Please enter your request [Q] : S
Choose key type:
  R - RSA key pair.
  D - DSA key pair.
  E - ECC key pair.
  Q - Quit.
Please enter your request [Q] : E
Using curve ec163a02. Generating key pair (please wait)...
Country: CA
State: Ontario
Locality: Waterloo
Organization: Sybase, Inc.
Organizational Unit: IAS
Common Name: IAS_Waterloo
Enter password to protect private key : mypwd123
Enter file path to save request : global.req
Enter file path to save private key : serv1_private_key.pri
```

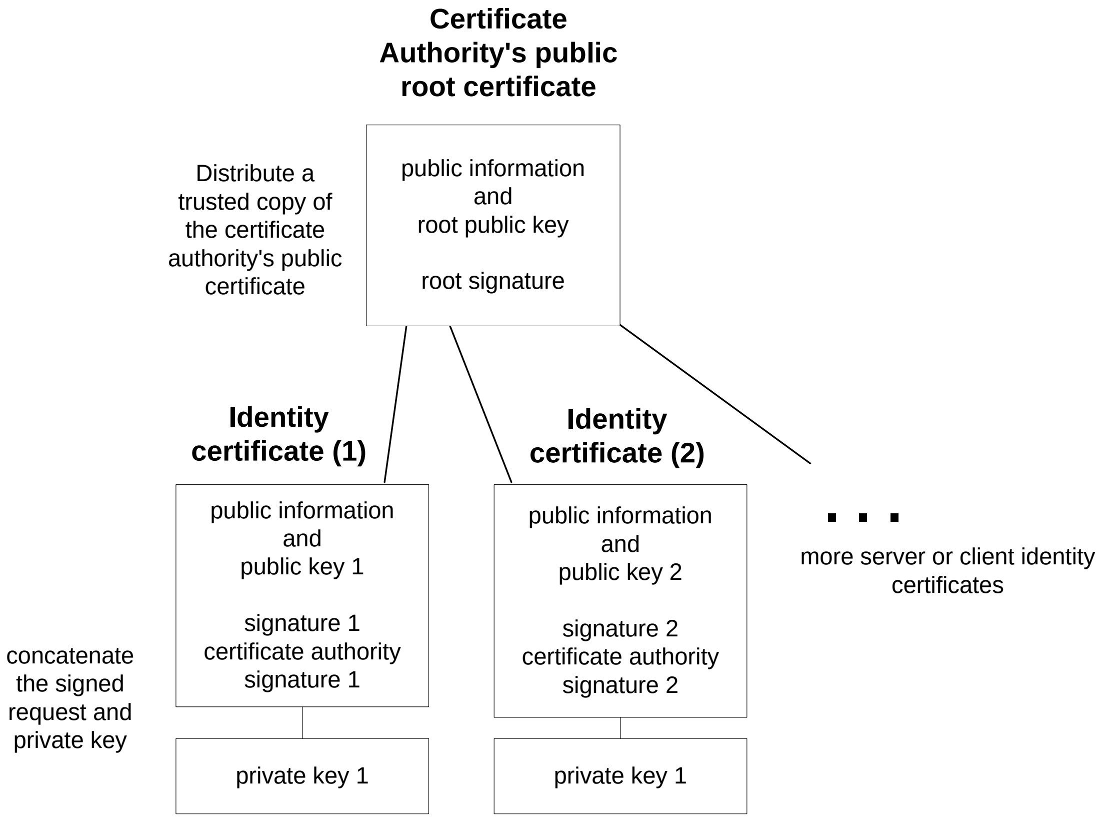
The file *global.req* contains the public certificate and request information. Paste the contents of this file into a form on the certificate-issuing web site. The Certificate Authority will sign the request and create the public certificate *global.crt*.

The file *serv1_private_key.pri* contains the corresponding private key. This file is protected by the password you entered, but since the protection provided by the password is weak, you must store this file in a secure location.

☞ For more information about using reqtool, see the document *reqtool.pdf*, located in the *win32* subdirectory of your SQL Anywhere 10 installation.

Using globally signed identity certificates

You can use globally-signed certificates directly as server identity certificates. The following diagram shows the configuration for multiple identity certificates:


Certificate Authority's public root certificate
public information
and
root public key
root signature
Distribute a trusted copy of the certificate authority's public certificate
Identity certificate (1)
public information
and
public key 1
signature 1
certificate authority signature 1
private key 1
Identity certificate (2)
public information
and
public key 2
signature 2
certificate authority signature 2
private key 1
. . .
more server or client identity certificates
concatenate the signed request and private key

To create the server identity, you must concatenate the public certificate signed by the Certificate Authority and private key created using the reqtool utility.

☞ For more information about the reqtool utility, see "Using reqtool to obtain global certificates" on page 895.

The following example concatenates the globally-signed public certificate *global.crt* and the private key *serv1_private_key.pri* to create the server certificate *server1_certificate.crt*.

```
copy global.crt+serv1_private_key.pri server1_certificate.crt
```

You reference the server certificate *server1_certificate.crt* and the password for the private key *serv1_private_key.pri* at the mlsrv10 or dbsrv10 command line.

☞ MobiLink: see “Starting the MobiLink server with transport-layer security” on page 904.

☞ SQL Anywhere: see “Starting the database server with transport-layer security” on page 898.

Setting up clients or servers to trust the certificate authority's public certificate

For server authentication, you must ensure that clients contacting your server trust the root certificate in the chain. In the case of globally-signed certificates, the root certificate is the Certificate Authority's public 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.

For more information about verifying certificate fields for globally-signed certificates, see “Verifying certificate fields” on page 906.

☞ For more information about configuring MobiLink clients to trust server certificates, see “Configuring MobiLink clients to use transport-layer security” on page 905.

☞ For more information about configuring the database server to use transport-layer security, see “Starting the database server with transport-layer security” on page 898.

☞ For more information about using globally-signed certificates to establish trust, see “Globally-signed certificates” on page 894.

Encrypting SQL Anywhere client/server communications

You can encrypt SQL Anywhere client/server communication using transport-layer security.

Starting the database server with transport-layer security

To start the database server with transport-layer security, supply the server certificate and the password protecting the server's private key. SQL Anywhere transport-layer security is only available over TCP/IP and on Solaris, Linux, NetWare, Mac OS X, or any supported Windows platform except Windows CE.

☞ For an overview of the steps required to set up transport-layer security, see "Setting up transport-layer security" on page 889.

Use the -ec server option to specify the certificate and certificate_password parameters.

Following is the syntax of a partial dbsrv10 command line:

```
-ec tls(
  tls_type=cipher;
  certificate=server-certificate;
  certificate_password=password)
-x tcpip
```

- ***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 the Separately licensed components table in SQL Anywhere 10.0.0 Components by Platform.

  The cipher must match the encryption (ECC or RSA) used to create your certificates.

  ☞ For information about enforcing the FIPS-approved algorithm, see "-fips server option" on page 144.

- ***server-certificate*** is the path and file name of the server 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 890.

- ***password*** is the password for the server certificate's private key. You specify this password when you create the server certificate.

You can also start the database server with simple encryption, which does not assure data integrity or provide server authentication. Simple encryption makes it more difficult for someone using a packet sniffer to read the network packets sent between the client and the server. Simple encryption is supported in previous versions of SQL Anywhere.

☞ For more information about the -ec server option, see "-ec server option" on page 139.

You specify the TCP/IP protocol using the -x server option.

☞ For more information, see "-x server option" on page 182.

Examples

The following example uses the -ec server option to specify ECC security, the server certificate, and the password protecting the server's private key:

```
dbsrv10 -ec tls(tls_type=ecc;certificate=c:\test\serv1_ecc.crt;
certificate_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. For more information, see "@data server option" on page 125.

The following example uses the -ec server option to specify RSA security, the server certificate, and the password protecting the server's private key:

```
dbsrv10 -ec tls(tls_type=rsa;certificate=c:\test\serv1_rsa.crt;
certificate_password=test) -x tcpip c:\test\secure.db
```

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 public 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 889.

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 public 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 891.

☞ For more information about enterprise root certificates and Certificate Authorities, see "Certificate chains" on page 892.

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 894.

When creating a certificate using the gencert utility, you enter values for the organization, organizational unit, and common name fields. You verify these fields using corresponding client connection parameters.

- **Organization** The organization field corresponds to the certificate_company encryption connection parameter.
- **Organizational unit** The organizational unit field corresponds to the certificate_unit encryption connection parameter.
- **Common name** The common name field corresponds to the certificate_name encryption connection parameter.

☞ For more information about client-side encryption connection parameters, see "Encryption connection parameter [ENC]" on page 222.

Client security options

Clients use a set of encryption connection parameters for transport-layer security.

Trusted_certificates option

This is the only required option. Clients use the trusted_certificates encryption option to specify trusted database server certificates. The trusted certificate can be a server's self-signed public certificate, a public enterprise root certificate, or a public certificate belonging to a commercial Certificate Authority.

☞ For more information, see "Creating digital certificates" on page 890.

Verifying certificate fields

The certificate_company, certificate_unit, and certificate_name encryption 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.

☞ For more information about verifying certificate fields, see "Verifying certificate fields" on page 900.

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)
```

- ♦ ***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 RSA FIPS cipher can only be used on supported 32-bit Windows platforms and Solaris.

  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 trusted public certificate. If you are using FIPS-approved RSA encryption, you must generate your certificates using RSA.

☞ For more information about trusted_certificates and other client security parameters, see "Client security options" on page 900.

☞ For more information about creating or obtaining the public certificate, see "Creating digital certificates" on page 890.

☞ For more information about the encryption connection parameter, see "Encryption connection parameter [ENC]" on page 222.

Example

The following example uses the trusted_certificates encryption connection parameter to specify the public certificate, *public_cert.crt*.

```
"UID=DBA;PWD=sql;ENG=myeng;LINKS=tcpip;
ENC=tls(tls_type=ecc;trusted_certificates=public_cert.crt)"
```

The following example uses the trusted_certificates encryption connection parameter to specify the public certificate, *public_cert.crt*, and verifies certificate fields using the certificate_unit and certificate_name encryption connection parameters.

```
"UID=DBA;PWD=sql;ENG=myeng;LINKS=tcpip;
ENC=tls(tls_type=ecc;trusted_certificates=public_cert.crt;
certificate_unit=test_unit;certificate_name=my_certificate)"
```

Using transport-layer security for SQL Anywhere web services

To set up transport-layer security for web services, perform the following steps:

- **Create digital certificates** You must create public certificates and server certificates. Public certificates (which can be Certificate Authority certificates) are distributed to browsers or web clients. Server certificates 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 890.

- **Start the web server with transport-layer security** Use the -xs database server option to specify HTTPS, the server certificate, and the password to protect the private key.

  Following is the syntax of a partial dbsrv10 command line.

  -xs *protocol***(**
  [**fips=**{ **y**|**n**};]
  Certificate=*server-certificate*;
  Certificate_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 server option" on page 144.

  - ***server-certificate*** The path and file name of the server certificate.

    For HTTPS, you must use an RSA certificate.

  - ***password*** The password for the server certificate's private key. You specify this password when you create the server certificate.

  ☞ For more information about the -xs server option, see "-xs server option" on page 185.

  ☞ For more information about the certificate and certificate_password parameters, see:

  - "Certificate protocol option" on page 244
  - "Certificate_Password protocol option" on page 244

- **Configure web clients** Configure browsers or other web clients to trust public 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 890.

Encrypting MobiLink client/server communications

You can encrypt MobiLink client/server communication using transport-layer security.

Starting the MobiLink server with transport-layer security

To start the MobiLink server with transport-layer security, supply the server certificate 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 889.

Securing the server over TCP/IP and HTTPS

Use the mlsrv10 -x server option to specify certificate and certificate_password parameters. Following is a partial mlsrv10 command line (which must be written on one line):

-x *protocol*(
  tls_type=*cipher***;**
  fips={ y | n };
  certificate=*server-certificate***;**
  certificate_password=*password***;***...***)**

- ♦ ***protocol*** can be **https** or **tls**. The **tls** protocol is TCP/IP with TLS.
- ♦ ***cipher*** can be **rsa** or **ecc** for RSA and ECC encryption, respectively. The cipher must match the encryption used to create your certificates.
- ♦ **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 CE.
- ♦ ***server-certificate*** is the path and file name of the server certificate.

  ☞ 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 890.
- ♦ ***password*** is the password for the server certificate's private key. You specify this password when you create the server certificate.

☞ For more information about the mlsrv10 -x option, see "-x option" [*MobiLink - Server Administration*].

Examples

The following example specifies the type of security (RSA), the server certificate, and the password protecting the server's private key at the mlsrv10 command line:

```
mlsrv10 -c "dsn=my_cons"
 -x tls(tls_type=rsa;certificate=c:\test
\serv_rsa1.crt;certificate_password=pwd)
```

The following example specifies an ECC certificate at the mlsrv10 command line:

```
mlsrv10 -c "dsn=my_cons"
 -x tls(tls_type=ecc;certificate=c:\test
\serv_ecc1.crt;certificate_password=pwd)
```

☞ For more information about the mlsrv10 -x option, see “-x option” [*MobiLink - Server Administration*].

☞ For more information about creating the server certificate, in this case *serv1.crt*, see “Creating digital certificates” on page 890.

☞ You can hide the command line options using a configuration file and the File Hiding utility (dbfhide). For more information, see “@data 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 public 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 889.

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 public 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 891.

☞ For more information about enterprise root certificates and Certificate Authorities, see "Certificate chains" on page 892.

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 894.

When creating a certificate using the gencert 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 909.
- "Client security options" on page 906.

☞ For more information about creating digital certificates, see "Creating digital certificates" on page 890.

Client security options

MobiLink clients (SQL Anywhere and UltraLite) use a common set of connection parameters to configure transport-layer security.

trusted_certificates parameter

MobiLink clients use the trusted_certificates connection parameter to specify trusted MobiLink server certificates. The trusted certificate can be a server's self-signed public certificate, a public enterprise root certificate, or the public certificate belonging to a commercial Certificate Authority.

☞ For more information, see:

- “trusted_certificates” [*MobiLink - Client Administration*]
- “Creating digital certificates” on page 890

Verifying certificate fields

The certificate_company, certificate_unit, and certificate_name connection parameters 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.

☞ For more information about verifying certificate fields, see:

- “Verifying certificate fields” on page 906
- “Globally-signed certificates” on page 894
- “Server authentication” on page 905

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.

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*** can be **https** or **tls**. The **tls** protocol is TCP/IP using transport-layer security.
- **fips** for FIPS-approved RSA encryption. FIPS-approved HTTPS uses separate FIPS 140-2 certified software from Certicom but is compatible with MobiLink servers using HTTPS and version 9.0.2 or later.
- ***public-certificate*** is the path and file name of a trusted public certificate.

  For HTTPS or FIPS-approved HTTPS, you must use certificates created using RSA encryption.

See also

☞ For more information about trusted_certificates and other client security parameters, see “Client security options” on page 906.

☞ For more information about creating or obtaining the public certificate, see “Creating digital certificates” on page 890.

☞ For more information about HTTPS parameters, see the HTTPS section in “CommunicationAddress (adr) extended option” [*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
 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=tcpip;
       adr=port=3333;
       tls(
           tls_type=rsa;
           trusted_certificates=c:\test\public_cert.crt;
           certificate_company=Sybase, Inc.;
           certificate_unit=IAS;
           certificate_name=MobiLink)"
```

Another option is to 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;
    security=tls(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" [*MobiLink - Client Administration*].

☞ 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** For information, see "UltraLite Create Database utility (ulcreate)" [*UltraLite - Database Management and Reference*] or "UltraLite Initialize Database utility (ulinit)" [*UltraLite - Database Management and Reference*].
   - **Using the trusted_certificates protocol option** For details, see step 3 of this procedure. This option is not available on Palm OS.

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_synch_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_synch_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 parameters are used to verify certificate fields.

   ☞ For more information about verifying certificate fields, see "Verifying certificate fields" on page 906.

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). The trusted certificate option is not available on Palm OS.

```
auto ul_synch_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" [*MobiLink - Client Administration*]

Certificate reader utility [readcert]

Use the readcert utility to display values within a certificate and validate the chain of certificates.

Syntax

readcert *certificate-name*

Description

The certificate you specify can be ECC or RSA.

When synchronization occurs through an ECC or RSA synchronization stream, the MobiLink server sends its certificate to the client, as well as the certificate of the entity that signed it, and so on up to a self-signed root. The client checks that the chain is valid and that it trusts the root certificate in the chain.

This utility scans an X.509 authentication certificate and displays the field values. It then checks that the chain of certificates is valid. A validation error is reported if any of the certificates in the chain have expired, are in the wrong order, or are missing.

See also

♦ "Certificate generation utility [gencert]" on page 912

Certificate generation utility [gencert]

Use the gencert utility to create new ECC or RSA certificates, or to sign pre-generated certificate requests.

Syntax

gencert [**-c** | **-s**] [**-r** | **-q**]

Option	Description
-c	Specifies a certificate you can use to sign other certificates. If used with -r, generates an enterprise root certificate.
-s	Specifies a server identity certificate. The server identity is a combination of a server's private key and public certificate. You reference the server identity certificate when you start the MobiLink server (for MobiLink transport-layer security) or database server (for SQL Anywhere client-server transport-layer security). If used with -r, generates a self-signed server certificate.
-r	Specifies a self-signed root certificate. If used with -s, gencert creates a self-signed server certificate. If used with -c, gencert creates an enterprise root certificate you can use to sign other certificates. If you specify gencert -r with no additional options, gencert creates a certificate you can use as a server certificate or an enterprise root. This option is not compatible with -q.
-q *request-file*	Sign a pre-generated certificate request. If used with -s, gencert creates a server certificate. If used with -c, gencert creates an enterprise root certificate you can use to sign other certificates. If you specify gencert -q with no additional options, gencert creates a certificate you can use as a server certificate or an enterprise root. The -q option is not compatible with -r.

If you do not specify -s or -c , the certificate contains the functionality provided by both options, so it can be used to sign other certificates or you can use it directly as a server certificate.

Description

You can use the gencert utility to generate trusted public certificates, private keys, and server certificates used to secure MobiLink synchronizations or SQL Anywhere client-server communication. This utility creates X509 certificates (a standard certificate format) for various security configurations.

Gencert prompts you for the following information:

- **Cipher** Gencert prompts you to choose an ECC or RSA cipher. If you are generating an ECC certificate, gencert generates an ECC key pair. If you are generating an RSA certificate, it prompts for a key size between 512 and 2048, and then creates a certificate using RSA. (In general, longer keys provide stronger encryption but take longer to process.)

Whichever cipher you choose, you must specify that cipher when you start the server and client. For ECC certificates, specify tls_type=ecc, and for RSA certificates, specify tls_type=rsa or tls_type=rsa;fips=yes.

- **Country, State/Province, and Locality** These values provide general certificate identification. The locality fields are also required by third-party Certificate Authorities if you plan to use globally-signed certificates.

  ☞ For more information about using Certificate Authorities, see "Globally-signed certificates" on page 894.

- **Organization, Organizational Unit, and Common Name** These fields provide additional security that the client is authenticating the correct certificate. On the client side, they correspond to the certificate_company, certificate_unit, and certificate_name protocol options, respectively.

  ☞ See "Verifying certificate fields" on page 906.

- **Serial number** You are prompted to choose a serial number for the certificate. The serial number must use alphanumeric characters.
- **Certificate valid for how many years** You are prompted for the period (in years) that the certificate remains valid. If the certificate expires, all certificates signed by this certificate will also be invalid. Following the specified period, you will need to regenerate the enterprise root, each server certificate, and the public certificates distributed to clients.
- **Enter password to protect private key** This is the password you will specify in the certificate_password protocol option.
- **Enter file path to save certificate** Choose a file name and location for the certificate.
- **Enter file path to save private key** Choose a file name and location for the private key.
- **Enter file path to save server identity** Choose a file name and location for the server certificate.

See also

- "Certificates" on page 887
- "Certificate reader utility [readcert]" on page 911
- "-ec server option" on page 139
- "Encryption connection parameter [ENC]" on page 222
- "FIPS-approved encryption technology" on page 887

Examples

- The following example uses the gencert -r option to generate an RSA self-signed server certificate.

```
> gencert -r
Certificate Generation Tool
Choose certificate type ((R)SA or (E)CC): R
Enter key length (512-2048): 1024
Generating key pair...
Country: CA
State/Province: ON
Locality: Waterloo
Organization: Sybase, Inc.
Organizational Unit: IAS
Common Name: MobiLink
```

```
Serial Number: 123
Certificate valid for how many years: 12
Enter password to protect private key: mypwd123
Enter file path to save certificate: public_cert.crt
Enter file path to save private key: server_key.pri
Enter file path to save server identity: server_cert.crt
```

You distribute the self-signed public certificate *public_cert.crt* to clients. The server certificate *server_cert.crt* includes the server certificate and private key and is stored with the MobiLink server (for MobiLink transport-layer security) or the database server (for SQL Anywhere client-server transport-layer security).

- The following example uses the gencert -r and -c options to generate an enterprise root certificate.

```
> gencert -r -c
Certificate Generation Tool
Choose certificate type ((R)SA or (E)CC): E
Generating key pair...
Country: CA
State/Province: ON
Locality: Waterloo
Organization: Sybase, Inc.
Organizational Unit: IAS
Common Name: MobiLink
Serial Number: 123
Certificate valid for how many years: 15
Enter password to protect private key: mypwd123
Enter file path to save certificate: public_root_cert.crt
Enter file path to save private key: root_key.pri
```

The public enterprise root certificate *public_root_cert.crt* is distributed to clients. Store the enterprise private key *root_key.pri* in a secure location. You can use the generated files to sign server certificates.

- The following example uses the gencert -s option to create a signed server certificate.

```
> gencert -s
Certificate Generation Tool
Choose certificate type ((R)SA or (E)CC): E
Generating key pair...
Country: CA
State/Province: ON
Locality: Waterloo
Organization: Sybase, Inc.
Organizational Unit: IAS
Common Name: MobiLink
Serial Number: 123
Certificate valid for how many years: 7
Enter file path of signer's certificate: public_root_cert.crt
Enter file path of signer's private key: root_key.pri
Enter password for signer's private key: mypwd123
Enter password to protect private key: SERV1privatekeypwd
Enter file path to save server identity: serv1_cert.crt
```

The enterprise root certificate created in the previous example is used as the signing certificate. You reference the server identity *serv1_cert.crt* and the password SERV1privatekeypwd when you start the MobiLink server or your database server.

- The following example uses the gencert option -q to sign a certificate request called *certreq.txt*.

```
>gencert -s -q certreq.txt
Certificate Generation Tool
Serial Number: 01
Certificate valid for how many years: 10
Enter file path of signer's certificate: rsaroot.crt
Enter file path of signer's private key: rsaroot.key
Enter password for signer's private key: test
Enter file path to save certificate: testcert.crt
Save entire chain (y/n): y
```

The certificate request is created using reqtool. For more information, see:

- “Globally-signed certificates” on page 894 for MobiLink transport-layer security
- “Globally-signed certificates” on page 894 for SQL Anywhere transport-layer security

Part VII. Replication

This section describes how to use SQL Anywhere as an Open Server, as well as how to replicate data with the Replication Server.

CHAPTER 25

SQL Anywhere as an Open Server

Contents

About this chapter

SQL Anywhere can appear to client applications as an Open Server. This feature enables Sybase Open Client applications to connect natively to SQL Anywhere databases.

This chapter describes how to use SQL Anywhere as an Open Server, and how to configure Open Client and SQL Anywhere to work together.

☞ For information on developing Open Client applications for use with SQL Anywhere, see "Sybase Open Client API" [*SQL Anywhere Server - Programming*].

Open Clients, Open Servers, and TDS

This chapter describes how SQL Anywhere fits into the Sybase Open Client/Open Server client/server architecture. This section describes the key concepts of this architecture, and provides the conceptual background for the rest of the chapter.

If you simply want to use a Sybase application with SQL Anywhere, you do not need to know any details of Open Client, 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.

Open Clients and Open Servers

SQL Anywhere and other members of the Adaptive Server family act as **Open Servers**. This means you can develop client applications using the **Open Client** libraries available from Sybase. Open Client includes both the Client Library (CT-Library) and the older DB-Library interfaces.

Tabular Data Stream

Open Clients and Open Servers 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 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. Open Client applications and other Sybase applications such as Replication Server and 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, as well as 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 Replication Server and OmniConnect to work with SQL Anywhere.

Replication Server support

The Open Server interface enables support for Sybase Replication Server: Replication Server connects through the Open Server interface, enabling SQL Anywhere databases to act as replicate sites in Replication Server installations.

For your database to act as a primary site in a Replication Server installation, you must also use the Replication Agent for Sybase SQL Anywhere, also called a **Log Transfer Manager**.

☞ For information on the Replication Agent, see "Replicating Data with Replication Server" on page 933.

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 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 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 (*dbsrv10.exe*) if you want to access an Open Server over a network. You can use the personal server (*dbeng10.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):

- **Open Client components** The Open Client libraries provide the network libraries your application needs to communicate via TDS if your application uses 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, both of which are available at: http://www.sybase.com/products/informationmanagement/softwaredeveloperkit/jconnect.
- **DSEdit** You need DSEdit, the directory services editor, to make server names available to your Open Client application. On Unix platforms, this utility is called sybinit.

  DSEdit is not included with SQL Anywhere, but is included with 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:

```
dbsrv10 -x tcpip,spx c:\mydata.db
dbsrv10 -x tcpip -n myserver c:\mydata.db
```

The first command uses both TCP/IP and SPX protocols, of which TCP/IP is available for use by Open Client applications. The second line uses only TCP/IP.

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 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:

```
dbsrv10 -x tcpip(ServerPort=2629) -n myserver c:\mydata.db
```

You may also need to supply an EngineName if more than one local database server is running, or if you want to connect to a network server.

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 details on setting up the client computer, see "Configuring Open Servers" on page 924.

Configuring Open Servers

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 PC 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 Open Client applications on your computer. When you use an 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 setup program 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.

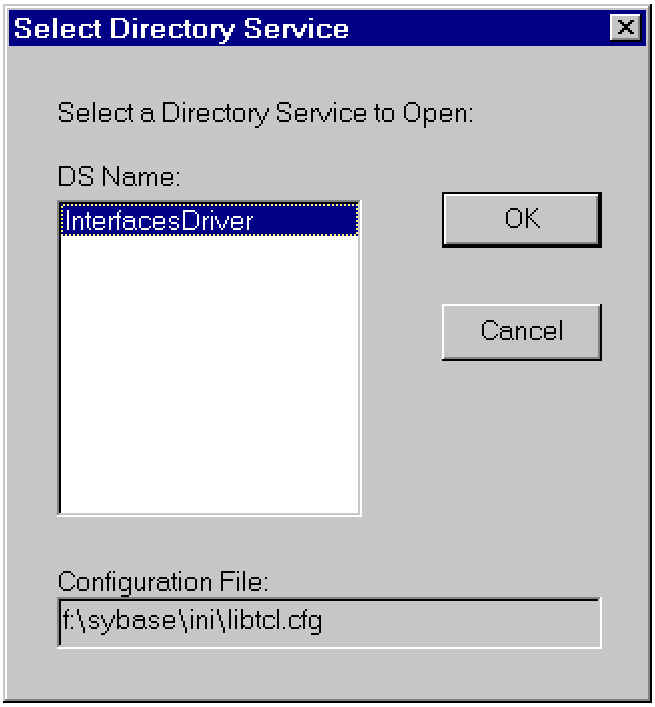
These sections describe how to use DSEdit for those tasks required for SQL Anywhere. It is not complete documentation for the DSEdit utility.

☞ For more information on DSEdit, see the *Utility Programs* book 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. You can start DSEdit either from a command prompt or from the Explorer in the standard fashion.

When you start DSEdit, the Select Directory Service dialog appears.



Opening a directory services session

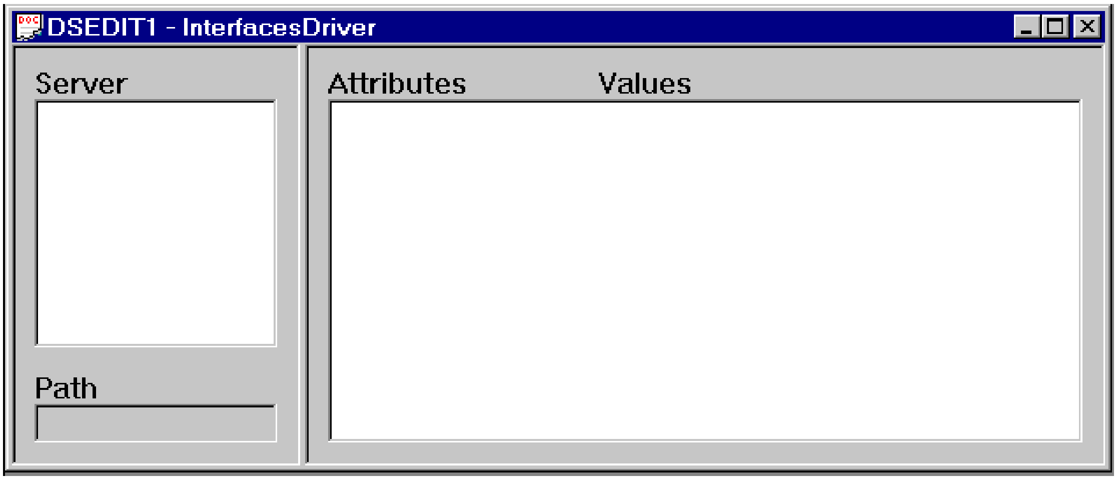
The Select Directory Service window allows you to open a session with a directory service. You can open a session to edit the interfaces file (*SQL.ini*), or any directory service that has a driver listed in the *libtcl.cfg* file.

♦ To open a session

- Click the local name of the directory service you want to connect to, as listed in the DS Name box, and click OK.

  For SQL Anywhere, select the 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 this window.

Adding a server entry

♦ To add a server entry

1. Choose Add from the Server Object menu.

   The Input Server Name window appears.

2. Enter a server name in the Server Name box, and click OK to enter the server name.

   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 Open Client. For SQL Anywhere, if the 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.

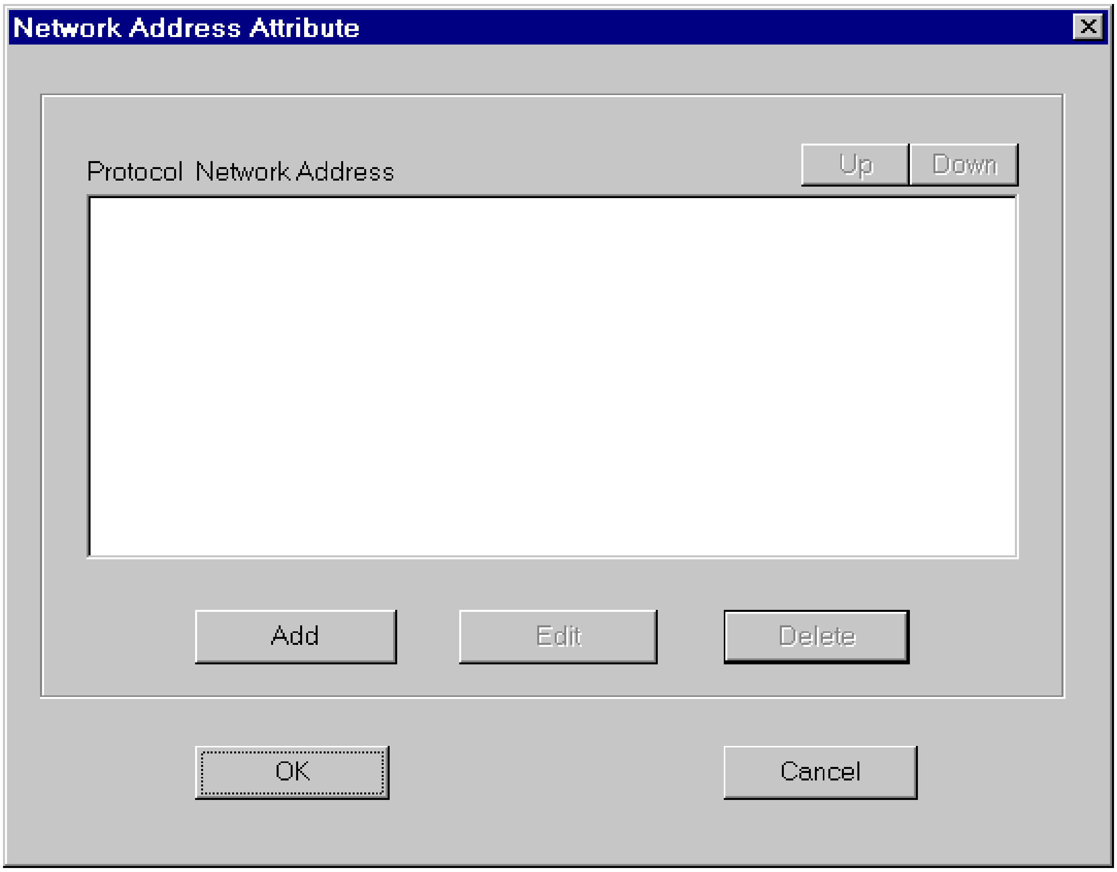
Adding or changing the server address

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. Select a server entry in the Server box.
2. Right-click the Server Address in the Attributes box.

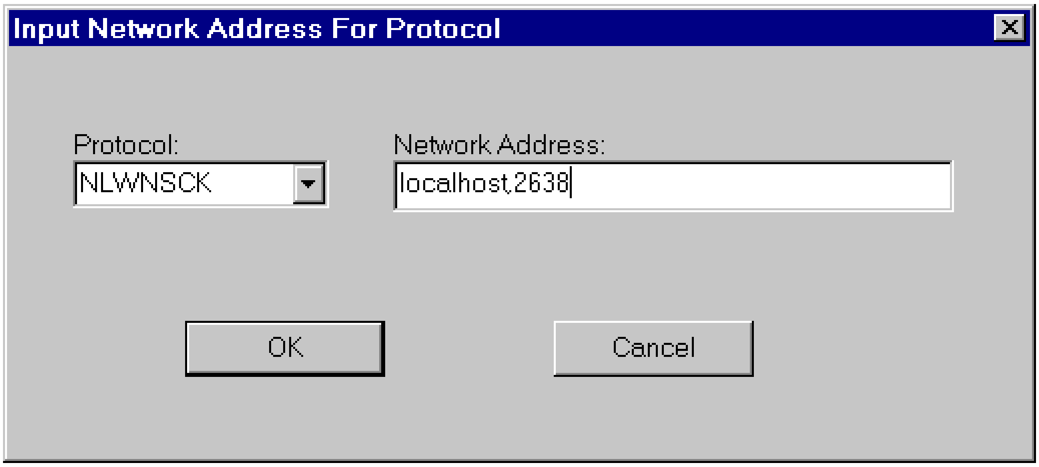
3. Choose Modify Attribute from the popup menu. A window appears, showing the current value of the address. If you have no address entered, the box is empty.



4. Click Add.

   The Input Network Address for Protocol window appears.

5. Select NLWNSCK from the Protocol dropdown list (this is the TCP/IP protocol) and enter a value in the Network Address field.


Input Network Address For Protocol

Protocol: NLWNSCK

Network Address: localhost,2638

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.

Machine 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 with, as described in "Starting the database server as an Open Server" on page 922. 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
```

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. Click the server entry in the Server box of the DSEdit session window.
3. Choose Ping Server from the Server Object menu.

   The Ping window appears.
4. Select the address you want to ping. Click Ping.

   A message box appears, notifying you whether or not the connection is successful. A message box 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. Select a server entry in the Server box.
2. Choose Rename from the Server Object menu.

   The Input Server Name window appears.
3. Enter a new name for the server entry in the Server Name box.
4. Click OK to make the change.

Deleting server entries

You can delete server entries from the DSEdit session window.

♦ To delete a server entry

1. Click a server entry in the Server box.
2. Choose Delete from the Server Object menu.

Configuring servers for JDBC

The JDBC connection address (URL) contains all the information required to locate the server.

☞ For information on the JDBC URL, see “Supplying a URL to the driver” [*SQL Anywhere Server - Programming*].

Characteristics of 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
automatic_timestamp	On
chained	Off
close_on_endtrans	Off
date_format	YYYY-MM-DD
date_order	MDY
escape_character	Off
float_as_double	On
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_hex_constant	On
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 automatic_timestamp='On';
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 float_as_double='On';
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_hex_constant='On';
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 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 371.

CHAPTER 26

Replicating Data with Replication Server

Contents

About this chapter

This chapter describes how you can use Sybase Replication Server to replicate data between a SQL Anywhere database and other databases. Other databases in the replication system may be SQL Anywhere databases or other kinds of database.

Before you begin

Replication Server administrators who are setting up SQL Anywhere to take part in their Replication Server installation will find this chapter especially useful. You should have knowledge of Replication Server documentation, and familiarity with the Replication Server product. This chapter does not describe Replication Server itself.

☞ For information about Replication Server, including design, commands, and administration, see your Replication Server documentation.

Note
SQL Anywhere includes components that allow you to use SQL Anywhere databases in a Replication Server system. Replication Server is not included as part of your SQL Anywhere installation.

Separately licensable option required
The Log Transfer Manager (LTM), which is the SQL Anywhere Replication Agent for Sybase Replication Server, is required for any SQL Anywhere database that participates in a Sybase Replication Server installation as a primary site. The license that is required for the LTM must be ordered separately. If SQL Anywhere is used as the replicate site, the LTM is not needed.
For more information, see "Separately licensed components" [*SQL Anywhere 10 - Introduction*].

Introduction to replication

Replication Server is a connection-based technology intended for the two-way replication of transactions. It is well suited to replication between a small number of enterprise databases connected by a high-speed network, generally with an administrator at each site. In such a setup, it is possible to achieve lag times as low as a few seconds.

You can also replicate SQL Anywhere data using MobiLink and SQL Remote.

☞ For information about SQL Remote, see SQL Remote [*SQL Remote*].

☞ For information about MobiLink, see “Introducing MobiLink Synchronization” [*MobiLink - Getting Started*].

Replication Server characteristics

Replication Server is designed for replication systems with the following requirements:

- **Small numbers of databases** Replication Server is designed to support replication among servers, with systems typically involving fewer than one hundred servers.
- **Continuously connected** Connections between primary sites and replicate sites may be over a wide area network, but Replication Server is designed for situations where there is a near-continuous connection path for data exchange among the servers in the system.
- **Low latency** Low latency means a short lag time between data being entered at one database and being replicated to each database in the system. With Replication Server, replication messages are sent typically within seconds of being entered at a primary site.
- **High volume** With near-continuous connections and high performance, Replication Server is designed for a high volume of replication messages.
- **Heterogeneous databases** Replication Server supports several leading DBMSs, and allows mapping of object names during replication, so that support for heterogeneous databases is provided.

Replicate sites and primary sites

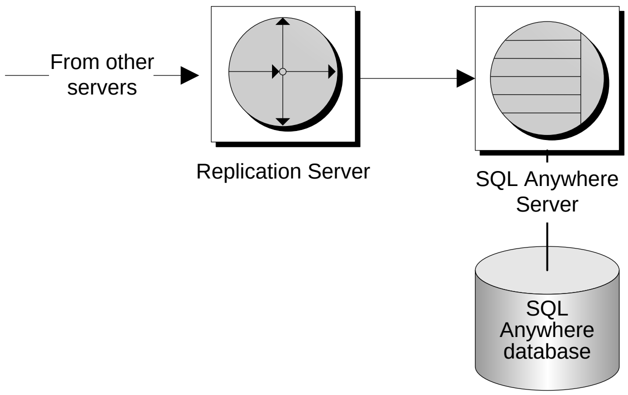
In a Replication Server installation, the data to be shared among databases is arranged in **replication subscriptions**.

For each replication definition, there is a **primary site**, where changes to the data in the replication occur. The sites that receive the data in the replication are called **replicate sites**.

Replicate site components

You can use SQL Anywhere as a replicate site. If you use SQL Anywhere as a replicate site, you do not need the LTM.

The following diagram illustrates the components required for SQL Anywhere to participate in a Replication Server installation as a replicate site.



- Replication Server receives data changes from primary site servers.
- Replication Server connects to SQL Anywhere to apply the changes.
- SQL Anywhere makes the changes to the database.

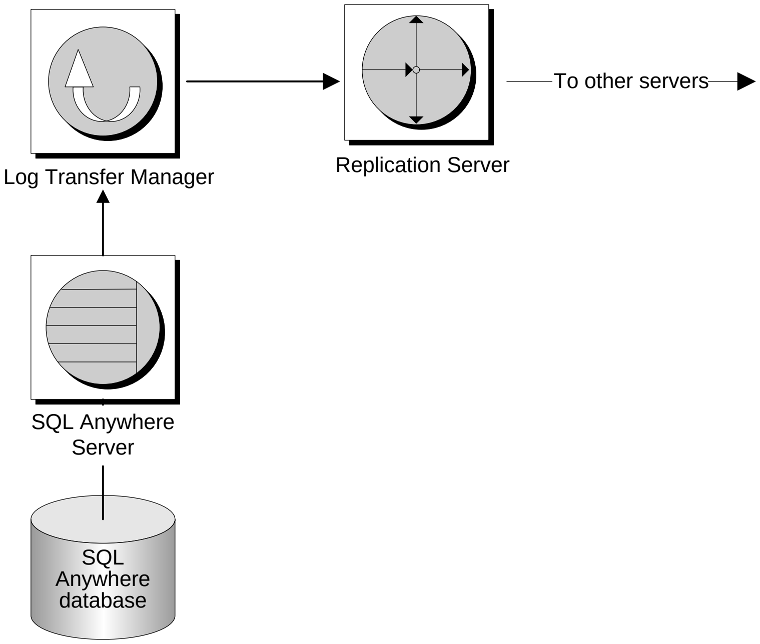
Asynchronous procedure calls

The Replication Server can use **asynchronous procedure calls** (APC) at replicate sites to alter data at a primary site database. If you are using APCs, the above diagram does not apply. Instead, the requirements are the same as for a primary site.

Primary site components

To use a SQL Anywhere database as a primary site, you need to use the Log Transfer Manager (LTM), the replication agent for SQL Anywhere. The LTM supports Replication Server version 10.0 and greater.

The following diagram illustrates the components required for SQL Anywhere to participate in a Replication Server installation as a primary site. The arrows in the diagram represent data flow.



- The SQL Anywhere database server manages the database.
- The SQL Anywhere Log Transfer Manager connects to the database. It scans the transaction log to pick up changes to the data, and sends them to Replication Server.
- Replication Server sends the changes to replicate site databases.

Tutorial: Replicate data using Replication Server

This section provides a step-by-step tutorial describing how to replicate data from a primary database to a replicate database. Both databases in the tutorial are SQL Anywhere databases.

Replication Server assumed

This section assumes you have a running Replication Server.

☞ For more information about how to install or configure Replication Server, see the Replication Server documentation.

What is in the tutorial

This tutorial describes how to replicate only tables.

☞ For information about replicating procedures, see “Preparing procedures and functions for replication” on page 950.

The tutorial uses a simple example of a (very) primitive office news system: a single table with an ID column holding an integer, a column holding the user ID of the author of the news item, and a column holding the text of the news item. The id column and the author column make up the primary key.

Before you work through the tutorial, create a directory (for example, *c:\tutorial*) to hold the files you create in the tutorial.

Lesson 1: Create the SQL Anywhere databases

This section describes how to create and set up the SQL Anywhere databases for replication.

You can create a database using Sybase Central or the dbinit utility. For this tutorial, you use the dbinit utility.

♦ Create the primary site database

- Enter the following command from the tutorial directory you created (for example, *c:\tutorial*).

  ```
  dbinit primedb
  ```

  This creates a database file *primedb.db* in the current directory.

♦ Create the replicate site database

- Enter the following command from the tutorial directory you created (for example *c:\tutorial*).

  ```
  dbinit repdb
  ```

  This creates a database file *repdb.db* in the current directory.

What's next?

Next, you have to start database servers to run the databases.

Lesson 2: Start the database servers

You need to run the primary site database server, with the primary database loaded.

♦ Start the primary site database server

1. Change to the tutorial directory.
2. Enter the following command to start a network database server running the primedb database. You should be using the TCP/IP network communication protocol on the default communications port (2638):

   ```
   dbsrv10 -x tcpip primedb.db
   ```

♦ Start the replicate site database server

1. Change to the tutorial directory.
2. Enter the following command to start a network database server running the repdb database, but on a different port:

   ```
   dbsrv10 -x tcpip(PORT=2639) -n REPSV repdb.db
   ```

What's next?

Next, you have to make entries for each of the SQL Anywhere servers in an interfaces file, so Replication Server can communicate with these database servers.

Lesson 3: Set up the Open Servers in your system

You need to add a set of Open Servers to the list of Open Servers in your system.

Adding Open Servers

Open Servers are defined in your interfaces file (*SQL.ini*) using the DSEdit utility. For Unix users, the interfaces file is named *interfaces*, and the utility is named sybinit.

☞ For full instructions on how to add definitions to your interfaces file, see "Configuring Open Servers" on page 924.

Required Open Servers

For each Open Server definition, you must provide a **name** and an **address**. Do not alter the other attributes of the definition. You need to add an Open Server entry for each of the following:

- **The primary database** Create an entry named PRIMEDB with address as follows:
  - **Protocol** NLWNSCK
  - **Network address** localhost,2638
- **The replicate database** Create an entry named REPDB with address as follows:

  - **Protocol** NLWNSCK
  - **Network address** localhost,2639

- **The LTM at the primary database** This is necessary so you can shut down the LTM properly. Create an entry named PRIMELTM with address as follows:

  - **Protocol** NLWNSCK
  - **Network address** localhost,2640

- **Your Replication Server** This tutorial assumes you already have the Replication Server Open Server defined.

What's next?

Next, confirm that the Open Servers are configured properly.

Lesson 4: Confirm that the Open Servers are configured properly

You can confirm that each Open Server is available by selecting ServerObject ► Ping Server from the DSEdit utility.

Alternatively, you can confirm that each Open Server is configured properly by connecting to the database using an Open Client application such as the isql utility.

To start isql running on the primary site database, type

```
isql -U DBA -P sql -S PRIMEDB
```

The Open Client isql utility is not the same as the SQL Anywhere Interactive SQL utility.

Lesson 5: Add Replication Server information to the primary database

You need to add Replication Server tables and procedures to the primary site database for the database to participate in a Replication Server installation. You also need to create two user IDs for use by Replication Server. The SQL command file *rssetup.sql* is included with SQL Anywhere and performs these tasks.

The *rssetup.sql* command file must be run on the SQL Anywhere server from the Interactive SQL utility.

♦ Run the rssetup script

1. From Interactive SQL, connect to the SQL Anywhere database as the DBA.
2. Run the rssetup script using the following command:

   ```
   read "install-dir\scripts\rssetup.sql"
   ```

   In this script, *install-dir* is your SQL Anywhere installation directory.

   Alternatively, you can choose File ► Run Script, and browse to the file.

Actions performed by rssetup.sql

The *rssetup.sql* command file performs the following functions:

- Creates a user named **dbmaint**, with password **dbmaint**, who has DBA permissions. This is the maintenance user name and password required by Replication Server to connect to the primary site database.

- Creates a user named **sa**, with password **sysadmin**, who has DBA permissions. This is the user ID used by Replication Server when materializing data.

- Adds **sa** and **dbmaint** to a group named **rs_systabgroup**.

Passwords and user IDs

While the hard-wired user IDs (dbmaint and sa) and passwords are useful for test and tutorial purposes, you should change the password and perhaps also the user IDs when running databases that require security. Users with DBA permissions have full authority in a SQL Anywhere database.

The user ID sa and its password must match that of the system administrator account on the Replication Server. SQL Anywhere does not currently accept a NULL password.

Permissions

The *rssetup.sql* script performs a number of operations, including some permissions management. The permissions changes made by *rssetup.sql* are outlined here. *You do not have to make these changes yourself.*

For replication, ensure that the dbmaint and sa users can access this table without explicitly specifying the owner. To do this, the table owner user ID must have group membership permissions, and the dbmaint and sa users must be members of the table owner group. To grant group permissions, you must have DBA authority.

For example, if the user DBA owns the table, you should grant group permissions to DBA:

```
GRANT GROUP
TO DBA
```

You should then grant the dbmaint and sa users membership in the DBA group. To grant group membership, you must either have DBA authority or be the group ID.

```
GRANT MEMBERSHIP
IN GROUP "DBA"
TO dbmaint ;
GRANT MEMBERSHIP
IN GROUP "DBA"
TO sa;
```

Lesson 6: Create the table for the primary database

In this section, you create a single table in the primary site database, using isql. First, make sure you are connected to the primary site database:

```
isql -U DBA -P sql -S PRIMEDB
```

Next, create a table in the database:

```
CREATE TABLE news (
  ID INT,
  AUTHOR CHAR( 40 ) DEFAULT CURRENT USER,
  TEXT CHAR( 255 ),
  PRIMARY KEY ( ID, AUTHOR )
)
go
```

Identifier case sensitivity

In SQL Anywhere, all identifiers are case insensitive. In Adaptive Server Enterprise, identifiers are case sensitive by default. Even in SQL Anywhere, ensure the case of your identifiers matches in all parts of the SQL statement to ensure compatibility with Adaptive Server Enterprise.

In SQL Anywhere, passwords are always case sensitive. User IDs, being identifiers, are case insensitive in all SQL Anywhere databases.

For more information, see "CREATE DATABASE statement" [*SQL Anywhere Server - SQL Reference*].

For news to act as part of a replication primary site, you must set REPLICATE to ON for the table using an ALTER TABLE statement:

```
ALTER TABLE news
REPLICATE ON
go
```

This is equivalent to running the sp_setreplicate or sp_setreptable procedure on the table in Adaptive Server Enterprise. You cannot set REPLICATE ON in a CREATE TABLE statement.

Lesson 7: Add Replication Server information to the replicate database

You should run the *rssetup.sql* command file on the replicate database in exactly the same manner as it ran on the primary database. Also ensure that the dbmaint and sa users can access this table without explicitly specifying the table owner.

☞ These tasks are the same as those performed on the primary database. For a complete explanation, see "Lesson 5: Add Replication Server information to the primary database" on page 939.

Lesson 8: Create the tables for the replicate database

The replicate site database needs to have tables to hold the data it receives. Now is a good time to create these tables. As long as the database elements are in place, no extra statements are necessary for them to act as a replicate site in a Replication Server installation. In particular, you do not need to set REPLICATE to ON, which is necessary only at the primary site.

Replication Server allows replication between tables and columns with different names. As a simple example, however, create a table in the replicate database identical in definition to that in the primary database (except for REPLICATE, which is not set to ON in the replicate database). The table creation statement for this is:

```
CREATE TABLE news (
   ID INT,
   AUTHOR CHAR( 40 ) DEFAULT CURRENT USER,
   TEXT CHAR( 255 ),
      PRIMARY KEY ( ID, AUTHOR )
)
go
```

For the tutorial, the CREATE TABLE statement must be *exactly* the same as that at the primary site.

You must ensure that the users dbmaint and sa can access this table without specifying the owner name. Also, these user IDs must have SELECT and UPDATE permissions on the table.

Lesson 9: Set up Replication Server

You need to perform the following tasks on the Replication Server:

- Create a connection for the primary site data server.
- Create a connection for the replicate site data server.
- Create a replication definition.
- Create a subscription to the replication.

This section describes each of these tasks. It also describes starting the SQL Anywhere LTM.

Create a connection for the primary site

Using isql, connect to Replication Server and create a connection to the primary site SQL Anywhere database.

The following command creates a connection to the primedb database on the PRIMEDB Open Server.

```
CREATE CONNECTION TO PRIMEDB.primedb
SET ERROR CLASS rs_sqlserver_error_class
SET FUNCTION STRING class rs_sqlserver_function_class
SET USERNAME dbmaint
SET PASSWORD dbmaint
WITH LOG TRANSFER ON
go
```

If you have changed the dbmaint user ID and password in the *rssetup.sql* command file, make sure you replace the dbmaint username and password in this command.

Replication Server does not actually use the primedb database name; instead, the database name is read from the command line of the PRIMEDB Open Server. You must, however, include a database name in the CREATE CONNECTION statement to conform to the syntax.

☞ For a full description of the create connection statement, see the chapter "Replication Server Commands" in *Replication Server Reference Manual*.

Create a connection for the replicate site.

Using isql, connect to Replication Server and create a connection to the replicate site SQL Anywhere database.

The following command creates a connection to the repdb database on the REPDB Open Server.

```
CREATE CONNECTION TO REPDB.repdb
SET ERROR CLASS rs_sqlserver_error_class
SET FUNCTION STRING CLASS rs_sqlserver_function_class
SET USERNAME dbmaint
SET PASSWORD dbmaint
go
```

This statement differs from the primary site server statement in that there is no WITH LOG TRANSFER ON clause in this statement.

If you have changed the dbmaint user ID and password in the *rssetup.sql* command file, make sure you replace the dbmaint username and password in this command.

Create a replication definition

Using isql, connect to Replication Server and create a replication definition. The following statement creates a replication definition for the news table on the primedb database:

```
CREATE REPLICATION DEFINITION NEWS
WITH PRIMARY AT PRIMEDB.primedb
( id INT, author CHAR(40), text CHAR(255) )
PRIMARY KEY ( id, author )
go
```

☞ For a full description of the CREATE REPLICATION DEFINITION statement, see *Replication Server Reference Manual*.

If you set the qualify_table_owners option to On in the LTM configuration file, you must specify the table owner in the statement, for all replicating tables.

Configure and start the SQL Anywhere LTM

For replication to take place, the SQL Anywhere LTM must be running against the primary site server. Before you start the SQL Anywhere LTM, make sure it is properly configured by editing an LTM configuration file.

Below is a sample configuration file for the primedb database. If you are following the examples, you should make a copy of this file as *primeltm.cfg*:

```
#
# Configuration file for 'PRIMELTM'
#
SQL_server=PRIMEDB
SQL_database=primedb
SQL_user=sa
SQL_pw=sysadmin
RS_source_ds=PRIMEDB
RS_source_db=primedb
RS=your_rep_server_name_here
RS_user=sa
RS_pw=sysadmin
LTM_admin_user=DBA
```

```
LTM_admin_pw=sql
LTM_charset=cp850
scan_retry=2
APC_user=sa
APC_pw=sysadmin
SQL_log_files=C:\TUTORIAL
```

If you have changed the user ID and password in the *rssetup.sql* command file for sa and sysadmin, you should use the new user ID and password in this configuration.

To start the SQL Anywhere LTM running on the primary site server, enter the following command:

```
dbltm -S PRIMELTM -C primeltm.cfg
```

The connection information is in *primeltm.cfg*. In this command, PRIMELTM is the server name of the LTM.

You can find usage information about the SQL Anywhere LTM by typing the following statement:

```
dbltm -?
```

You can run the SQL Anywhere LTM as a Windows service.

☞ For information on running programs as services, see "Running the server outside the current session" on page 33.

Create a subscription for your replication

Using isql, connect to Replication Server and create a subscription for the replication.

The following statement creates a subscription for the news replication defined in "Create a replication definition" on page 943 with the replicate site as the repdb database.

```
CREATE SUBSCRIPTION NEWS_SUBSCRIPTION
FOR news
WITH REPLICATE AT REPDB.repdb
go
```

You have now completed your installation. Try replicating data to confirm that the setup is working properly.

Lesson 10: Enter data at the primary site for replication

You can now replicate data from the primary database to the replicate database. As an example, connect to the primary database using the isql utility, and enter a row in the news table.

```
INSERT news (id, text)
VALUES (1, 'Test news item.' )
COMMIT
go
```

The SQL Anywhere LTM sends only committed changes to the Replication Server. The data change is replicated next time the LTM polls the transaction log.

Tutorial complete

You have now completed the tutorial.

Configuring databases for Replication Server

Each SQL Anywhere database that participates in a Replication Server installation needs to be configured before it can do so. Configuring the database involves the following tasks:

- Selecting a secure user ID for the maintenance user and the name used by Replication Server when materializing data.
- Setting up the database for Replication Server.
- Configuring the language and character set, where necessary.

Configuring the LTM

Each primary site SQL Anywhere database requires an LTM to send data to Replication Server. Each primary or replicate site SQL Anywhere database requires an Open Server definition so that Replication Server can connect to the database.

☞ For information on configuring the LTM, see "Configuring the LTM" on page 952.

Setting up the database for Replication Server

Once you have created your SQL Anywhere database and created the necessary tables and so on within the database, you can make database ready for use with Replication Server. You do this using a setup script supplied with the SQL Anywhere Replication Agent product. The script is named *rssetup.sql*.

When you need to run the setup script

You need to run the setup script at any SQL Anywhere database that is taking part in a Replication Server installation, whether as a primary or a replicate site.

What the setup script does

The setup script creates user IDs required by Replication Server when connecting to the database. It also creates a set of stored procedures and tables used by Replication Server. The tables begin with the characters **rs_**, and the procedures begin with the characters **sp_**. Procedures include some that are important for character set and language configuration.

Prepare to run the setup script

Replication Server uses a special data server **maintenance user** login name for each local database containing replicated tables. This allows Replication Server to maintain and update the replicated tables in the database.

The maintenance user

The setup script creates a maintenance user with name **dbmaint** and password **dbmaint**. The maintenance user has DBA permissions in the SQL Anywhere database, which allows it full control over the database. For security reasons, you should change the maintenance user ID and password.

♦ To change the maintenance user ID and password

1. Open the *rssetup.sql* setup script in a text editor. The script is held in the *scripts* subdirectory of your SQL Anywhere installation directory.
2. Change all occurrences of the dbmaint user ID to the new maintenance user ID of your choice.
3. Change the dbmaint password to the new maintenance user password of your choice. The password occurs in the following place at the top of the setup script file:

   ```
   GRANT CONNECT TO dbmaint
   IDENTIFIED BY dbmaint;
   ```

The materialization user ID

When Replication Server connects to a database to materialize the initial copy of the data in the replication, it does so using the Replication Server system administrator account.

The SQL Anywhere database must have a user ID and password that match the Replication Server system administrator user ID and password. SQL Anywhere does not accept a NULL password.

The setup script assumes a user ID of sa and a password of **sysadmin** for the Replication Server administrator. You should change this to match the actual name and password.

♦ To change the system administrator user ID and password

1. Open the *rssetup.sql* setup script in a text editor.
2. Change all occurrences of the sa user ID to match the Replication Server system administrator user ID.
3. Change the sa user's password to match the Replication Server system administrator password.

   The password has the initial setting of **sysadmin**.

Run the setup script

Once you have modified the setup script to match the user IDs and passwords appropriately, you can run the setup script to create the maintenance and system administrator users in the SQL Anywhere database.

♦ To run the setup script

1. Start the SQL Anywhere database on a SQL Anywhere database server.
2. Start the Interactive SQL utility, and connect to the database as the DBA.

   When you create a SQL Anywhere database, it contains the user ID **DBA** with password **sql**. This user has DBA authority.
3. Run the script by entering the following command in the SQL Statements pane:

   ```
   read install-dir\scripts\rssetup.sql
   ```

   In this command, *install-dir* is your SQL Anywhere installation directory.

Replication Server character set and language issues

Upon creation, each SQL Anywhere database is assigned a specific collation (character set and sort order). Replication Server uses a different set of identifiers for character sets and sort orders.

Set the character set and language parameters in the LTM configuration file. If you are unsure of the character set label to specify, you can do the following to determine the character set of the server:

♦ To determine the character set

- Execute the following command:

  ```
  exec sp_serverinfo csname
  ```

The language is one of the language labels listed in "Language label values" on page 315.

Identifiers in Replication Server

If you are using the SQL Anywhere Replication Agent with Replication Server 15.0 and Open Client/Open Server 15.0, the Replication Agent supports table, column, procedure, function, and parameter names up to 128 bytes in length.

The maximum length for identifiers when using the Replication Agent with older versions of Replication Server and Open Client/Open Server is 30 bytes.

Using the LTM

Since the SQL Anywhere LTM relies on information in the SQL Anywhere transaction log, take care not to delete or damage the log without storing backups (for example, using a transaction log mirror).

☞ For more information about transaction log management, see the section “Transaction log and backup management” on page 955.

You cannot substitute a SQL Anywhere LTM for an Adaptive Server Enterprise LTM since the transaction logs have different formats.

The SQL Anywhere LTM supports replication of inserts, updates, and deletes, as well as replication of Transact-SQL dialect stored procedure calls.

The Adaptive Server Enterprise LTM sends data changes to the Replication Server before they are committed. The Replication Server holds the changes until a COMMIT statement arrives. By contrast, the SQL Anywhere LTM sends only committed changes to Replication Server. For long transactions, this may lead to some added delay in replication, since all changes have to go through the Replication Server before distribution.

Configuring tables for replication

You can use sp_setreplicate or sp_setrepproc system procedure or the ALTER TABLE statement to configure tables for replication. A table is identified as a primary data source using the ALTER TABLE statement with a single clause:

```
ALTER TABLE table-name
SET REPLICATE ON;
```

The effects of setting REPLICATE ON for a table

Setting REPLICATE ON places extra information into the transaction log. Whenever an UPDATE, INSERT, or DELETE action occurs on the table. The SQL Anywhere Replication Agent uses this extra information to submit the full pre-image of the row, where required, to Replication Server for replication.

Even if only some of the data in the table needs to be replicated, all changes to the table are submitted to Replication Server. It is Replication Server's responsibility to distinguish the data to be replicated from that which is not.

When you update, insert, or delete a row, the pre-image of the row is the contents of the row before the action, and the post-image is the contents of the row after the action. For INSERTS, only the post-image is submitted (the pre-image is empty). For DELETES, the post-image is empty and only the pre-image is submitted. For UPDATES, both the pre-image and the updated values are submitted.

The following data types are supported for replication:

Data type	Description (Open Client/Open Server type)
Exact integer data types	int, smallint, tinyint

Data type	Description (Open Client/Open Server type)
Exact decimal data types	decimal, numeric
Approximate numeric data types	float (8-byte), real
Money data types	money, smallmoney
Character data types	char(n), varchar(n), text
Date and time data types	datetime, smalldatetime
Binary data types	binary(n), varbinary(n), image
Bit data types	bit

Notes

SQL Anywhere supports data of zero length that is not NULL. However, non-null long varchar and long binary data of zero length is replicated to a replicate site as NULL.

If a primary table has columns with unsupported data types, you can replicate the data if you create a replication definition using a compatible supported data type. For example, to replicate a DOUBLE column, you could define the column as FLOAT in the replication definition.

Side effects of setting REPLICATE ON for a table

There can be a replication performance hit for heavily updated tables. You could consider using replicated procedures if you experience performance problems that may be related to replication traffic, since replicated procedures send only the call to the procedure instead of each individual action.

Since setting REPLICATE ON sends extra information to the transaction log, this log grows faster than for a non-replicating database.

Minimal column replication definitions

The SQL Anywhere LTM supports the Replication Server replicate minimal columns feature. This feature is enabled at Replication Server.

☞ For more information on replicate minimal columns, see your Replication Server documentation.

Preparing procedures and functions for replication

You can use stored procedures to modify the data in tables. Updates, inserts, and deletes execute from within the procedure.

Replication Server can replicate procedures as long as they satisfy certain conditions. The first statement in a procedure must perform an update for the procedure to be replicated.

☞ For a full description of how Replication Server replicates procedures, see your Replication Server documentation.

SQL Anywhere supports two dialects for stored procedures: the Watcom-SQL dialect, based on the draft ISO/ANSI standard, and the Transact-SQL dialect. You must use Transact-SQL to write stored procedures for replication.

Function APC format

The SQL Anywhere LTM supports the Replication Server **function APC** format. To make use of these functions, set the configuration parameter **rep_func** to **on** (the default is **off**).

The LTM interprets all replicated APCs as either table APCs or function APCs. A single SQL Anywhere database cannot combine function APCs with other table APCs.

☞ For more information about replicate functions, see your Replication Server documentation.

SQL statements for controlling procedure replication

A procedure can be configured to act as a replication source using the ALTER PROCEDURE statement.

The following statement makes the procedure MyProc act as a replication source.

```
ALTER PROCEDURE MyProc
REPLICATE ON;
```

The following statement prevents the procedure MyProc from acting as a replication source.

```
ALTER PROCEDURE MyProc
REPLICATE OFF;
```

You can also use the sp_setreplicate or sp_setrepproc system procedures to set up procedures for replication.

The effects of setting REPLICATE ON for a procedure

When a procedure is used as a replication data source, calling the procedure sends extra information to the transaction log.

Asynchronous procedures

A procedure called at a replicate site database to update data at a primary site database is an **asynchronous procedure**. The procedure performs no action at the replicate site, but rather, the call to the procedure is replicated to the primary site, where a procedure of the same name executes. This is called an **asynchronous procedure call** (APC). The changes made by the APC are then replicated from the primary to the replicate database in the usual manner.

☞ For information about APCs, see your Replication Server documentation.

The APC_user and APC support

Support for APCs in SQL Anywhere is different from that in Adaptive Server Enterprise. In Adaptive Server Enterprise, each APC executes using the user ID and password of the user who called the procedure at the replicate site. In SQL Anywhere, however, the transaction log does not store the password, and so it is not available at the primary site. To work around this difference, the LTM configuration file holds a single user ID with associated password, and this user ID (the **APC_user**) executes the procedure at the primary site.

The APC_user must, therefore, have appropriate permissions at the primary site for each APC that may be called.

Configuring the LTM

You control LTM behavior by modifying the LTM **configuration file**, which is a plain text file created and edited using a text editor. The LTM configuration file contains information the LTM needs, such as the SQL Anywhere server it transfers a log from, the Replication Server it transfers the log to. You need a valid configuration file to run the LTM.

Creating a configuration file

You must create a configuration file, using a text editor, before you can run the LTM. The -C LTM command specifies the name of the configuration file to use, and has a default of *dbltm.cfg*.

Configuration file format

The LTM configuration file shares the same format as the Replication Server configuration file described in your *Replication Server Administration Guide*. In summary:

- The configuration file contains one entry per line.
- An entry consists of a parameter, followed by the = character, followed by the value:

  ```
  Entry=value
  ```

- Lines beginning with a # character are comments ignored by the LTM.
- The configuration file cannot contain leading blanks.
- Entries are case sensitive.

☞ For the full list of available configuration file parameters, see "The LTM configuration file" on page 633.

Example configuration file

- The following is a sample SQL Anywhere LTM configuration file.

  ```
  # This is a comment line
  # Names are case sensitive.
  SQL_user=sa
  SQL_pw=sysadmin
  SQL_server=PRIMESV
  SQL_database=primedb
  RS_source_ds=PRIMESV
  RS_source_db=primedb
  RS=MY_REPSERVER
  RS_user=sa
  RS_pw=sysadmin
  LTM_admin_user=DBA
  LTM_admin_pw=sql
  LTM_charset=cp850
  scan_retry=2
  SQL_log_files=e:\logs\backup
  ```

```
APC_user=sa
APC_pw=sysadmin
```

Replicating transactions in batches

Effects of buffering transactions

The LTM allows buffering of replication commands to Replication Server. Buffering the replication commands and sending them in batches results in fewer messages being sent, and can significantly increase overall throughput, especially on high volume installations.

How batch mode works

By default, the LTM buffers transactions. The buffer flushes (the transactions sent to Replication Server when the buffer:

- **Reaches maximum number of commands** The **batch_ltl_sz** parameter sets the maximum number of LTL (log transfer language) commands stored in the buffer before it flushes. The default setting is 200.
- **Reaches maximum memory used** The **batch_ltl_mem** parameter sets the maximum memory that the buffer can occupy before flushes. The default setting is 256 KB.
- **Completes transaction log processing** If there are no more entries in the transaction log to process (that is, the LTM is up to date with all committed transactions), then the buffer flushes.

Turning off buffering

You can turn off buffering of transactions by setting the **batch_ltl_cmds** parameter to **off**:

```
batch_ltl_cmds=off
```

Language and character set issues

Language and character set issues are an important consideration in many replication sites. Each database and server in the system uses a specific collation (character set and sorting order) for storing and ordering strings. SQL Anywhere character set support is performed in a different manner to character set support in Adaptive Server Enterprise and other Open Client/Open Server based applications.

This section describes how to configure the SQL Anywhere LTM such that data in a SQL Anywhere database can be shared with Replication Server and hence with other databases.

The LTM automatically uses the default Open Client/Open Server language, sort order, and character set. You can override these defaults by adding entries to the LTM configuration file.

Open Client/Open Server collations

Adaptive Server Enterprise, Replication Server, and other Open Client/Open Server applications share a common means of managing character sets.

☞ For information on Open Client/Open Server character set support, see the chapter "Configuring Character Sets, Sort Orders, and Languages" in the Adaptive Server Enterprise *System Administration Guide*.

☞ For more information about character set issues in Replication Server, see the chapter "International Replication Design Considerations" in the *Replication Server Design Guide*.

This section provides a brief overview of Open Client/Open Server character set support.

Internationalization files

Files that support data processing in a particular language are called **internationalization files**. Several types of internationalization files come with Adaptive Server Enterprise and other Open Client/Open Server applications.

There is a directory named *charsets* under your Sybase directory. *Charsets* has a set of subdirectories, including one for each character set available to you. Each character set contains a set of files, as described in the following table

File	Description
Charset.loc	Character set definition files that define the lexical properties of each character such as alphanumeric, punctuation, operand, upper or lower case.
**.srt*	Defines the sort order for alphanumeric and special characters.
**.xlt*	Terminal-specific character translation files for use with utilities.

Character set settings in the LTM configuration file

Three settings in the LTM configuration file refer to character set issues:

- **LTM_charset** The character set for the LTM to use. You can specify any Sybase-supported character set.
- **LTM_language** The **language** used by the LTM to print its messages to the error log and to its clients. You can choose any language to which the LTM has been localized, as long as it is compatible with the LTM character set.

  The SQL Anywhere LTM has been localized to several languages.

Notes

Character set In an Open Client/Open Server environment, an LTM should use the same character set as the data server and Replication Server attached to it.

SQL Anywhere character sets are specified differently than Open Client/Open Server character sets. Consequently, the requirement is that the SQL Anywhere character set be compatible with the LTM character set.

Language The *locales.dat* file in the *locales* subdirectory of the Sybase release directory contains valid map settings. However, the LTM output messages in the user interface are currently available in those languages to which the LTM has been localized.

Sort order All sort orders in your replication system should be the same. You can find the default entry for your platform in the *locales.dat* file in the *locales* subdirectory of the Sybase release directory.

Example

♦ The following settings are valid for a Japanese installation:

```
LTM_charset=SJIS
LTM_language=Japanese
```

Transaction log and backup management

One of the differences between the Adaptive Server Enterprise LTM and the SQL Anywhere LTM is that while the Adaptive Server Enterprise LTM depends on a temporary recovery database for access to old transactions, the SQL Anywhere LTM depends on access to old transaction logs. No temporary recovery database exists for the SQL Anywhere LTM.

Replication depends on access to operations in the transaction log, and for SQL Anywhere primary site databases, sometimes access to old transaction logs. This section describes how to set up backup procedures at a SQL Anywhere primary site to ensure proper access to old transaction logs.

Consequences of lost transaction logs

Good backup practices at SQL Anywhere primary database sites are crucial. A lost transaction log could mean rematerializing replicate site databases. At primary database sites, a transaction log mirror is recommended.

☞ For information on transaction log mirrors and other backup procedure information, see “Backup and Data Recovery” on page 761.

The LTM configuration file contains a directory entry, which points to the directory where backed up transaction logs are kept. This section describes how to set up a backup procedure to ensure that such a directory stays in proper shape.

Backup utility options

With the Backup utility, you have the option of renaming the transaction log on backup and restart. For the dbbackup utility, this is the -r option. It is recommended that you use this option when backing up the primary database and replication database transaction logs.

For example, consider a database named *primedb.db*, in directory *c:\prime*, with a transaction log in directory *d:\primelog\primedb.log*. Backing up this transaction log to a directory *e:\primebak* using the rename and restart option performs the following tasks:

1. Backs up the transaction log, creating a backup file *e:\primebak\primedb.log*.
2. Renames the existing transaction log to *d:\primelog\YYMMDDxx.log*, where **xx** are sequential characters ranging from **AA** to **ZZ**.

3. Starts a new transaction log, as *d:\primelog\primedb.log*.

After several backups, the directory *d:\primelog* will contain a set of sequential transaction logs. The log directory should not contain any transaction logs other than the sequence of logs generated by this backup procedure.

Using the delete_old_logs option

The SQL Anywhere database delete_old_logs option is set to Off by default. If you set the default to on, the LTM automatically deletes the old transaction logs when Replication Server no longer needs access to the transactions. This option can help to manage disk space in replication setups.

For example, set the delete_old_logs option for the PUBLIC group:

```
SET OPTION PUBLIC.delete_old_logs = 'ON'
```

or

```
SET OPTION PUBLIC.delete_old_logs = '10 days'
```

☞ For more information, see “delete_old_logs option [MobiLink client] [SQL Remote] [Replication Agent]” on page 412.

The Unload utility and replication

If a database participates in replication, care must be taken when unloading and reloading to avoid needing to re-materialize the database. Replication is based on the transaction log, and unloading and reloading a database deletes the old transaction log. For this reason, good backup practices are especially important when participating in replication.

Replicating an entire database

SQL Anywhere provides a shortcut for replicating an entire database, so you don't have to set each table in the database as a replicated table.

You can set a PUBLIC database option called replicate_all using the SET OPTION statement. You can designate a whole database for replication using the following command:

```
SET OPTION PUBLIC.replicate_all='On';
```

You require DBA authority to change this and other PUBLIC option settings. You must restart the database for the new setting to take effect. The replicate_all option has no effect on procedures.

☞ For more information, see “replicate_all option [Replication Agent]” on page 450.

Stopping the LTM

You can shut down the LTM from the user interface in Windows XP/200x, or in other circumstances by issuing a command.

♦ To stop the LTM on Windows XP/200x when the LTM is not running as a service

- Click Shutdown on the user interface.

♦ To stop the LTM by issuing a command

1. Connect to the LTM from isql using the LTM_admin_user login name and password in the LTM configuration file. The user ID and password are case sensitive.
2. Stop the LTM using the SHUTDOWN statement.

Example

The following statements connect isql to the LTM PRIMELTM, and shut it down:

```
isql -SPRIMELTM -UDBA -Psql
1> shutdown
2> go
```

Part VIII. SQL Anywhere for Windows CE

This section describes how to use SQL Anywhere on Windows CE.

CHAPTER 27

SQL Anywhere for Windows CE

Contents

About this chapter

This chapter provides information about using SQL Anywhere on Windows CE. The information contained in this chapter is intended to supplement information contained throughout the remainder of the documentation.

☞ For information about security when running on Windows CE, see "Keeping your Windows CE database secure" on page 883.

Installing SQL Anywhere on a Windows CE device

Requirements

- Microsoft ActiveSync 3.5 or higher.
- A Windows CE device supported by SQL Anywhere.

  For a list of Windows CE devices supported by SQL Anywhere, see http://www.ianywhere.com/products/supported_platforms.html.
- A computer running a supported Windows operating system.

Windows CE file locations

The location of your SQL Anywhere install on Windows CE depends on the type of device and location you are installing to. No subdirectories are created. All DLLs are installed into the *\Windows* directory.

Operating system	Location	Installation directory
Windows CE 4.x	Main memory	*\Program Files\SQLAny10*
Windows CE 4.x	Storage card or SD card	*\storage-card\Sybase SQL Anywhere 10* *\SD-card\Sybase SQL Anywhere 10*
Windows CE 5.0	Main memory	*\Program Files\SQLAny10*
Windows CE 5.0	Storage card or SD card	*\storage-card\SQLAny10* *\SD-card\SQLAny10*

Installation considerations: using ICU on Windows CE

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 CE, you require ICU if UCA is used as the NCHAR collation or the CHAR collation because the UCA requires ICU. You also require ICU on Windows CE if your CHAR character set does not match your operating system character set.

ICU is not used for charset conversion on CE, so you must use either the operating system character set or UTF-8 for Windows CE databases. If you do not install the ICU libraries, then you must choose UTF8BIN as the NCHAR collation when creating your database. See “Installation considerations: using ICU on Windows CE” on page 962.

By default, the ICU libraries are not installed on Windows CE because they add approximately 1.7 MB to the size of the SQL Anywhere installation on Windows CE. However, you can modify your SQL Anywhere installation at a later date if you require these libraries.

Creating databases on the desktop to deploy to Windows CE
When creating a database on the desktop to deploy to a Windows CE device, you must ensure that you only use the UCA collation if the ICU libraries are installed on the Windows CE device. A database that uses the UCA is unusable on Windows CE if the ICU libraries are not installed on the device.

☞ For more information about ICU, see "Unicode Collation Algorithm (UCA)" on page 318, and "Understanding character set conversion" on page 322.

Installation considerations: using the .NET Compact Framework on Windows CE

Although ADO.NET 2.0 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 on a device, you must download the support for ADO.NET 2.0 from Microsoft, and install it to your device.

As part of the SQL Anywhere installation, you must specify which version of the .NET Compact Framework you want to use if you plan to develop ADO.NET applications:

- **Version 1.x** This is the default. For information about developing an application with ADO.NET 1.0, refer to the API reference for the SQL Anywhere .NET Data Provider in the *Adaptive Server Anywhere Programming Guide* at http://www.ianywhere.com/developer/product_manuals/sqlanywhere/0902/en/html/index.html.
- **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 "iAnywhere.Data.SQLAnywhere namespace (.NET 2.0)" [*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: deploying SQL Anywhere 10 on Windows Mobile 5

Windows Mobile 5 uses digital code signing to verify the origins of digital content. When deploying SQL Anywhere to a Windows Mobile 5 device, you can deploy either of the following types of *.cab* files:

- **Pre-built, signed .cab file** The SQL Anywhere installation includes pre-built *.cab* files that are signed by VeriSign. When you use these *.cab* files, you do not receive an unknown publisher warning. Files are supplied for both version 1.1 and 2.0 of the .NET Compact Framework.
- **Custom, unsigned .cab file** You can build a custom, unsigned *.cab* file to deploy SQL Anywhere to any supported Windows CE platform. On Windows Mobile 5, running a custom, unsigned *.cab* file results in an unknown publisher warning, but does not affect the installation of the software.

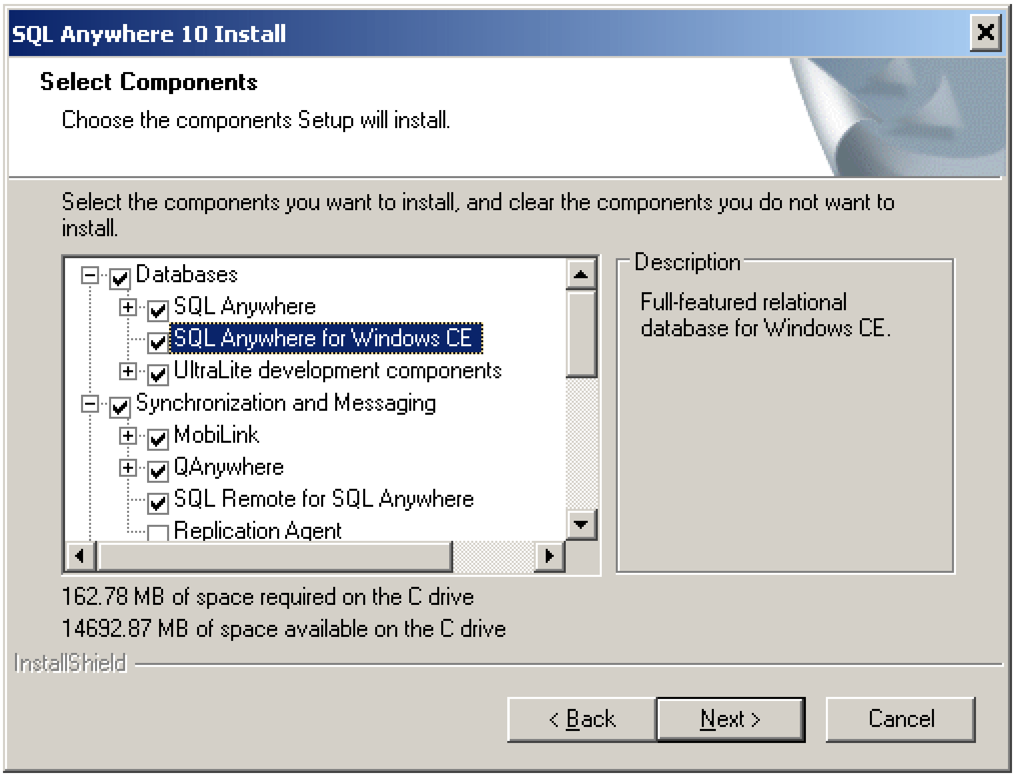
Installing SQL Anywhere for Windows CE

The following procedure takes you through the steps to install SQL Anywhere for Windows CE on your Windows CE device.

♦ To install SQL Anywhere for Windows CE

1. Connect your Windows CE device to a computer running a supported Windows operating system.
2. Start the installation by running the SQL Anywhere *setup.exe* program on your computer.
3. Follow the instructions in the SQL Anywhere 10 Install wizard.
4. On the Select Components page, select the SQL Anywhere for Windows CE option under Databases.

   If you already have this version of SQL Anywhere installed on your computer, clear the SQL Anywhere checkbox under Databases to avoid reinstalling the software on your computer.



SQL Anywhere 10 Install

Select Components

Choose the components Setup will install.

Select the components you want to install, and clear the components you do not want to install.

- ☑ Databases
  - ☑ SQL Anywhere
  - ☑ SQL Anywhere for Windows CE
  - ☑ UltraLite development components
- ☑ Synchronization and Messaging
  - ☑ MobiLink
  - ☑ QAnywhere
  - ☑ SQL Remote for SQL Anywhere
  - □ Replication Agent

Description

Full-featured relational database for Windows CE.

162.78 MB of space required on the C drive

14692.87 MB of space available on the C drive

5. Follow the remaining instructions in the wizard.

Using the Windows CE sample applications

This section introduces you to the sample database and the sample applications provided with SQL Anywhere for Windows CE.

The sample database represents a small company that makes a limited range of sports clothing.

The sample database is stored in a file called *demo.db*. On Windows CE devices, this file is located in the *\Program Files\SQLAny10* directory. There are two versions of the sample database available for Windows CE: one that includes the International Components for Unicode (ICU) libraries, and one that does not. SQL Anywhere implements character set conversion using the ICU open source library, developed and maintained by IBM.

☞ For more information about ICU and Windows CE, see "Installation considerations: using ICU on Windows CE" on page 962.

The following sample applications are included with your SQL Anywhere for Windows CE installation:

- ADOCE Sample
- ADO.NET Sample
- ESQL Sample
- ODBC Sample
- SQL Anywhere Server Example

You can use these applications to access the sample database and examine the capabilities of SQL Anywhere for Windows CE.

The SQL Anywhere Server Example

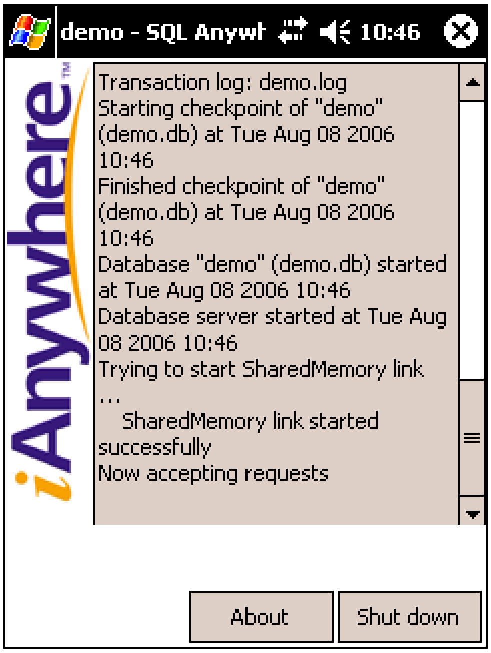
The SQL Anywhere Server Example starts the sample database on a network database server using preset server options and connection parameters.

♦ To start the SQL Anywhere Server Example

1. On your Windows CE device, navigate to the SQL Anywhere installation directory by tapping Start ► Programs ► SQLAny10.

2. Start the sample database on a network database server by tapping SQL Anywhere Server Example.

   The sample database starts running on the network database server. Once it starts, the database server appears as an icon in the bottom right corner of the Today screen on your Windows CE device. You can view the Server Messages window by tapping this icon.



```
Transaction log: demo.log
Starting checkpoint of "demo"
(demo.db) at Tue Aug 08 2006
10:46
Finished checkpoint of "demo"
(demo.db) at Tue Aug 08 2006
10:46
Database "demo" (demo.db) started
at Tue Aug 08 2006 10:46
Database server started at Tue Aug
08 2006 10:46
Trying to start SharedMemory link
...
   SharedMemory link started
successfully
Now accepting requests
```

You can now connect to the sample database on your Windows CE device from a computer.

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.

   The Server Messages window appears.

2. Tap Shut Down.

The ADO.NET Sample

To use the ADO.NET Sample, you must have the Microsoft .NET Compact Framework version 1.0 with Service Pack 2 or higher installed on your device.

You can download this component from the Microsoft Download Center at 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 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. Start the ADO.NET Sample by tapping Start ▶ Programs ▶ SQLAny10 ▶ ADO.NET Sample.

2. Tap Connect.

   The sample database starts on a network database server. The Server Messages window appears briefly, then disappears.

3. Tap Exec SQL to execute the default SQL statement, `SELECT * FROM Employees`, and then tap Exec SQL.

   All the 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.

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` in the Exec SQL field 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. Shut down the database server by tapping Disconnect.

The ADO.NET Sample disconnects, and the database server automatically shuts down.

14. Close the ADO.NET Sample by tapping the × in the top right corner of the window.

The ADOCE Sample

The ADOCE Sample demonstrates a simple application that uses ADOCE, the ADO programming interface for Windows CE development. This application allows you to start the sample database running on the network database server, and access data using SQL statements.

You must have Visual Basic installed on your Windows CE device to use the ADOCE sample.

The source code for this sample can be found in on your computer in *samples-dir\SQLAnywhere\ce\adoce_sample*.

You can load this project in eMbedded Visual Basic from the following location on your computer: *samples-dir\SQLAnywhere\ce\adoce_sample*.

Note
In the ADOCE Sample user interface, SQL statements must be entered on a single line.

♦ To use the ADOCE Sample

1. Start the ADOCE Sample by tapping Start ▶ Programs ▶ SQLAny10.
2. Tap ADOCE Sample.
3. Tap Connect.

   The sample database starts on a network database server. The Server Messages window appears briefly, then disappears.
4. Tap Execute SQL to execute the default SQL statement, `SELECT * FROM Employees`.

   All the data from the Employees table appears in the data window.
5. Navigate through the data in the Employees table using the scroll bar on the side of the data window.
6. Access a more specific range of data by typing `SELECT EmployeeID, Surname FROM Employees` and then tapping Execute SQL to execute the statement.
7. Scroll down to view the data, which appears below the data that is already displayed in the data window.
8. Access data from another table by typing `SELECT * FROM Customers` in the Execute SQL field and then tapping Execute SQL.
9. Scroll down to view the data from the Customers table, which appears below the data that is already displayed in the data window.
10. Shut down the network database server by tapping Disconnect.

    The ADOCE Sample disconnects, and the network database server automatically shuts down.

11. Close the ADOCE Sample by tapping the × 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 ► SQLAny10.
2. Tap ESQL Sample.
3. Tap Connect to connect to the sample database using the default connection string.

   The sample database starts on a network database server. The Server Messages window appears briefly, then disappears.
4. Tap ExecSQL to execute the default SQL statement, `SELECT * FROM Employees`.

   All the data from the Employees table appears in the data window.
5. Navigate through the data in the Employees table using the scroll bars on the bottom and side of the data window.
6. Access data in another table by typing `SELECT * FROM Customers` in the ExecSQL field, and then tapping ExecSQL.

   All the data from the Customers table appears in the data window, replacing the data from the Employees table.
7. Shut down the network database server by tapping Disconnect.

   The ESQL Sample disconnects and the network database server shuts down.
8. Close the ESQL Sample by tapping OK in the bottom left 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 ► SQLAny10 ► ODBC Sample.
2. Tap Connect.

   The sample database starts on a network database server. The Server Messages window appears briefly, then disappears.
3. Tap ExecSQL to execute the default SQL statement, `SELECT * FROM Employees`.

   All the 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 bottom and side of the data window.
5. Access data in another table by typing `SELECT * FROM Customers` in the ExecSQL field and then tapping ExecSQL.

   All the data from the Customers table appears in the data window, replacing the data from the Employees table.
6. Shut down the network database server by tapping Disconnect.

   The ODBC Sample disconnects and the network database server automatically shuts down.
7. Close the ODBC Sample by tapping OK in the bottom left corner of the window.

Connecting to a database running on a Windows CE device

If you want to connect an application running on a computer to a database running on a Windows CE device, you can connect over TCP/IP over the ActiveSync link between the computer and the Windows CE device. This allows you to administer a Windows CE database using the administration utilities on the computer.

☞ For more information about Windows CE administration, see “Using the administration utilities on Windows CE” on page 986.

Determining the IP address of your Windows CE device

When connecting to a database that is running on Windows CE, you may need the IP address to establish the connection.

If you have an ActiveSync connection via USB, such as when the device is cradled, you must set up a proxy port to be able to connect to the Windows CE database.

☞ For information about configuring proxy ports for Windows CE, see “Creating proxy ports for Windows CE devices” on page 972.

♦ To determine the IP address of your Windows CE device

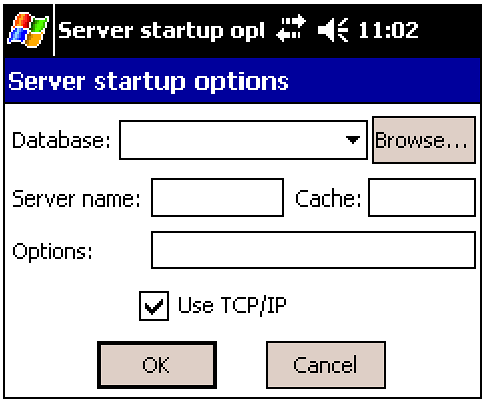
1. Open File Explorer on your Windows CE device.

   From the Start menu, tap Programs ► File Explorer.

2. Navigate to the SQL Anywhere directory by tapping Program Files ► SQLAny10.

3. Tap the *dbsrv10* icon.

   The Server Startup Options dialog appears.



Server startup options

Database:

Server name:

Cache:

Options:

☑ Use TCP/IP

4. Specify the database file that you want to start.

   Type **/Program Files/SQLAny10/demo.db** to start the sample database running on the device.

5. Select Use TCP/IP.

   A TCP/IP connection is necessary to connect from a computer to the database running on your Windows CE device.

6. In the Options field of the Server Startup Options dialog, type **-z**.

   With the -z option, the server writes out its IP address during startup. The address may change if you disconnect your Windows CE device from the network and then re-connect it.

   ☞ For more information, see "-z server option" on page 187.

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 Server icon located in the bottom right corner of the screen.

   The Server Messages window appears. The IP address appears in the Server Messages window.

Now you can create an ODBC data source to connect from the computer to your Windows CE device.

☞ For more information, see "Creating an ODBC data source to connect to your Windows CE device" on page 973.

Creating proxy ports for Windows CE devices

When you run a database on your Windows CE device, you must configure a proxy port for Active Sync so that requests are passed down to the database running on the Windows CE device.

♦ To add a proxy port for a Windows CE device (Interactive SQL)

1. Open Interactive SQL.

2. From the SQL menu, choose Connect.

   The Connect dialog appears.

3. Click Tools and then choose Setup Windows CE Proxy Port.

   The Windows CE Proxy Ports dialog appears.

4. Click New.

   The New Proxy Port dialog appears.

5. Type **SQL Anywhere** in the name field.

6. Type **2638** in the Port field.

This is the default TCP/IP port for SQL Anywhere.

> **Note**
> Every time your Windows CE device is cradled, ActiveSync forwards traffic on port 2638 to the device. If you start a database server on your computer while your Windows CE device is cradled, it cannot use the default port 2638. If this is problematic, you can choose another port to dedicate to Windows CE traffic.

You can also configure a proxy port by setting the following registry key: H*KEY_LOCAL_MACHINE \SOFTWARE\Microsoft\Windows CE Services\ProxyPorts*. Name the DWORD value SA and specify the port dedicated to Windows CE traffic (2638 by default).

> **Caution**
> Modifying the registry is dangerous. Modify the registry at your own risk.

Creating an ODBC data source to connect to your Windows CE device

This section describes how to create an ODBC data source on your Windows computer that connects to a database running on your Windows CE device.

☞ For more information about ODBC data sources, see "Working with ODBC data sources" on page 64.

♦ To create an ODBC data source to connect to your Windows CE device

1. Open the ODBC Administrator on the computer.

   From the Start menu, choose Programs ► SQL Anywhere 10 ► SQL Anywhere ► ODBC Administrator.

   The ODBC Administrator appears.

2. On the User DSN tab, click Add.

   The Create New Data Source dialog appears.

3. Select SQL Anywhere 10, and then click Finish.

   The ODBC Configuration dialog appears.

4. On the ODBC tab, in the Data Source Name field, type a name for the data source.

   For example, type **CEdevice**.

5. On the Login tab, select Supply User ID and Password. Make sure the User ID and Password fields blank.

   Each time you connect to a database, you must supply a user ID and password.

6. On the Database tab, leave the Server Name field blank.

Each time you connect from a computer, you must specify the server name. This name appears in the title bar of the Server Messages window on your Windows CE device.

7. On the Network tab, select the TCP/IP checkbox.

8. In the adjacent field, type the connection parameters.

   For example, type **Host=127.0.0.1;DoBroadcast=none**.

   ♦ **Host** specifies the IP address that the Windows CE device listens on.

     If you created a proxy port to connect to your Windows CE device, use the default IP address of **127.0.0.1** (this is the default localhost address).

     For more information, see "Creating proxy ports for Windows CE devices" on page 972.

     Otherwise, use the IP address of your Windows CE device.

     For more information, see "Determining the IP address of your Windows CE device" on page 971.

   ♦ **DoBroadcast** controls how the TCP/IP connection is made.

     When **DoBroadcast=none** is specified, the TCP/IP connection is made directly with the port specified. Use this setting if you created a proxy port to connect to your Windows CE device.

     For more information, see "Creating proxy ports for Windows CE devices" on page 972.

     When **DoBroadcast=direct** is specified, no broadcast is performed to the local subnet to search for a database server. Instead, the Host IP address is required.

     For more information, see "DoBroadcast protocol option [DOBROAD]" on page 247.

9. Click OK to create the data source.

You can now use this data source to connect from a computer to a database running on your Windows CE device.

☞ For more information, see "Using the administration utilities on Windows CE" on page 986.

Configuring Windows CE databases

Most SQL features available in the full version of SQL Anywhere are also available in the Windows CE 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 CE.

You should be mindful of the unsupported features when setting database properties on a database intended for a Windows CE device.

☞ For a complete list of unsupported features, see “SQL Anywhere feature support on Windows CE” on page 994.

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 CE, 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 317.

  Because character set translation is not supported on Windows CE, you must use either the operating system character set or UTF-8 for Windows CE databases.

  You must choose whether you want to install the ICU libraries when creating your Windows CE database. See “Installation considerations: using ICU on Windows CE” on page 962.

Using a transaction log on Windows CE

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 CE device.

When you copy an existing database to your Windows CE 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 CE 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 CE device.

☞ For more information about transaction logs, see "The transaction log" on page 766.

Using jConnect on Windows CE

jConnect is a pure Java JDBC driver for SQL Anywhere. Sybase Central and Interactive SQL give you the option to enable the jConnect JDBC driver so that Java applications can access SQL Anywhere databases.

By default, jConnect is not enabled for databases being created for Windows CE. However, you can choose to enable jConnect if you require it.

Adding jConnect 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 CE, it is recommended that you do not jConnect support to your database.

☞ For more information about using jConnect, see "Using the jConnect JDBC driver" [*SQL Anywhere Server - Programming*].

Using encryption on Windows CE

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.

See also

- "Encrypting a database" on page 873
- "Keeping your Windows CE database secure" on page 883

Creating a Windows CE database

There are three methods for creating a SQL Anywhere database for your Windows CE device:

- With the Sybase Central Create Database wizard to create a database that can be copied directly to your Windows CE device.
- With the Initialization utility (dbinit) to create a database that can be copied manually to your Windows CE device.
- With the CREATE DATABASE statement in Interactive SQL to create a database that can be copied manually to your Windows CE device.

Enable checksums for Windows CE databases
When creating a database that will be deployed to Windows CE, you should enable checksums. This helps to provide early detection if the database file becomes corrupt.

For information about decisions you need to make creating a Windows CE database, see:

- “Using a transaction log on Windows CE” on page 975
- “Installation considerations: using ICU on Windows CE” on page 962
- “Installation considerations: using the .NET Compact Framework on Windows CE” on page 963

Creating a Windows CE database using Sybase Central

Sybase Central has features to make database creation easy for Windows CE. If you have Windows CE services installed on your Windows computer, you can create a Windows CE database using Sybase Central. Sybase Central enforces the requirements for Windows CE databases, and gives you the option of copying the database file to your Windows CE device.

Enable checksums for Windows CE databases
When creating a database that will be deployed to Windows CE, you should enable checksums. This helps to provide early detection if the database file becomes corrupt.

♦ To create a Windows CE database in Sybase Central and copy it directly to your Windows CE device

1. Ensure your Windows CE device is connected to your computer.

2. Start Sybase Central on your computer: from the Start menu, choose Programs ► SQL Anywhere 10 ► Sybase Central.

3. Start the Create Database wizard: from the Tools menu, choose SQL Anywhere 10 ► Create Database.

   The Create Database wizard appears.

4. Choose Create a Database on This Computer, and then click Next.

   The database is initialized on the computer before it is copied to the Windows CE device.

5. Specify a file name and directory to store the database file in on your computer, and then click Next.

6. Select Create This Database for Windows CE, and then click Next.

   This option ensures that the new database conforms to the settings required to run the database on Windows CE.

   When you click Next, Sybase Central attempts to connect to your Windows CE device.

7. Select Copy the Database to Your Windows CE Device to have Sybase Central automatically copy the database to your device once it is initialized. Click Next.

8. Specify the Windows CE 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 CE device to make it simpler to start the database.
   When starting a database on your Windows CE device using the Server Startup Options dialog, 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 dialog.

   Optionally, you can select the Delete the Desktop Database After Copying option.

   If you choose not to delete the computer copy, a copy of the database file is stored on your computer in the directory that you specified in Step 5. Click Next.

9. On the Specify a Collation Sequence for NCHAR Data page, select the collation for NCHAR data for your database. If you are not using character set conversion and do not require the ICU libraries, select UTF8BIN.

10. Specify the directory where you want to save the transaction log file. Click Next.

    On your Windows CE 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.

11. Specify whether you want to use a transaction log mirror. Click Next.

    A mirror log can help in data recovery should the transaction log become corrupt. However, it is not necessary.

12. Ensure the Install jConnect Meta-information Support option is not selected. Click Next.

13. Set the level of encryption for your database by selecting the appropriate option. 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.

    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.

14. Ensure that the Include Checksum With Each Database Page option is selected. You can select the other options on this page as required for your database. Click Next.

    Follow the remaining instructions in the wizard.

15. Click Finish to create the database and copy it to your device.

    A dialog appears, tracking the progress of the files being copied to your Windows CE device.

16. Once the wizard has copied the database to your Windows CE device, you can verify the location of the files.

    From the Start menu, tap Programs ► File Explorer and navigate to the Windows CE 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 CE device.

Creating a Windows CE database using dbinit

The Initialization utility (dbinit) can be used to create databases that can be used on Windows CE. However, you cannot copy them directly to a Windows CE device from this utility. You must manually copy databases created with the dbinit utility to your Windows CE device.

Enable checksums for Windows CE databases
When creating a database that will be deployed to Windows CE, you should enable checksums. This helps to provide early detection if the database file becomes corrupt.

♦ To create a database using the dbinit utility

1. At a command prompt on your computer, navigate to the directory where you want to create your database.

   For example:

   ```
   cd temp
   ```

2. Create your database by entering 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 614.

3. Copy the database to your Windows CE device.

   ☞ For more about copying the database to your Windows CE device, see “Copying a database to your Windows CE device” on page 980.

Creating a Windows CE database using the CREATE DATABASE statement

The CREATE DATABASE statement can be used to create databases in Interactive SQL on your computer. However, you cannot copy them directly to a Windows CE device from this application. You must manually copy databases to your Windows CE device.

Enable checksums for Windows CE databases

When creating a database that will be deployed to Windows CE, you should enable checksums. This helps to provide early detection if the database file becomes corrupt.

♦ To create a database using the CREATE DATABASE statement

1. Open Interactive SQL on your computer.

   From the Start menu, choose Programs ▶ SQL Anywhere 10 ▶ Interactive SQL.

2. The Connect dialog appears.

   If the Connect dialog does not appear automatically, choose SQL ▶ Connect.

3. On the Identification tab, select the ODBC Data Source Name option, and type **SQL Anywhere 10 Demo** in the adjacent field.

4. Click OK to connect to the sample database.

5. 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.

   For more information, see “CREATE DATABASE statement” [*SQL Anywhere Server - SQL Reference*].

6. From the SQL menu, choose Execute.

   A database and transaction log are created in the *c:\temp* directory of your computer.

   ☞ For information about copying the database to your Windows CE device, see “Copying a database to your Windows CE device” on page 980.

Copying a database to your Windows CE device

Any existing SQL Anywhere database can be copied to your Windows CE device using the method described in this section. However, you must keep in mind that any database features that are not supported on Windows CE will not work when you copy the database to your Windows CE device. See “SQL Anywhere feature support on Windows CE” on page 994.

♦ To copy a database to your Windows CE device

1. Connect the Windows CE device to your computer.

2. Open Windows Explorer on your computer.

3. Browse to the directory on your computer containing the database that you want to copy.
4. Right-click the database file and choose Copy.
5. Open a second instance of Windows Explorer.
6. Browse to the directory on your Windows CE device where you want to store the database file.

> **Tip**
> When starting a database on your Windows CE device using the Server Startup Options dialog, 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 dialog.

7. Right-click an open area of the Windows Explorer window for your Windows CE device and then choose Paste.

   The file is copied to the Windows CE device.

Rebuilding databases on Windows CE

When rebuilding a database on Windows CE, you have the following options:

- Rebuild the Windows CE database on another platform and then copy the database to the Windows CE device. This is the recommended method for rebuilding a Windows CE database.
- Repopulate an empty database using dbmlsync.
- Repopulate an empty database using dbremote.
- Use dbunload on the Windows CE device.

The first three options are recommended when upgrading a Windows CE database. However, if these options are not available to you, you can use dbunload on CE. Before deciding to use dbunload on Windows CE, you should consider the following implications of using dbunload on Windows CE:

- 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 CE device

Because running dbunload on a Windows CE 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 CE device, you must choose the Unload/Reload Support option in the Deploy SQL Anywhere 10 for Windows CE 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 CE.

Notes about using dbunload on Windows CE

To use dbunload on a Windows CE device, ensure you have performed the following tasks:

- The following files should be deployed to your SQL Anywhere installation directory (by default, *\Program Files\SQLAny10*):
  - *dbsrv10.exe*
  - *dbunlspt.exe*
  - *dbunload.exe*
  - *dbrunsql.exe*
  - *unloadold.sql*
  - *unload.sql*
  - *optdeflt.sql*
  - *opttemp.sql*
- The following files should be deployed to the *\Windows* directory:
  - *dblgen10.dll*
  - *dblib10.dll*
  - *dbtool10.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\10.0\Location*.

The following steps can be embedded into third-party Windows CE 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 *dbtool10.dll*.

♦ To unload a database on Windows CE (dbunload)

1. On a platform other than Windows CE, create a new, empty SQL Anywhere 10 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 (this way, *dbicu10.dll* is not required by the database server).

2. Copy the SQL Anywhere 10 software and the empty SQL Anywhere 10 database file to the Windows CE device. See "Notes about using dbunload on Windows CE" on page 982.

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. Execute the following command:

```
dbrunsql-path\dbrunsql -c "UID=DBA;PWD=sql;CHARSET=none;DBF=new-empty-SQLAnywhere10databasefile" -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 CE device.

Backing up a Windows CE 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 CE database to a physically separate location to safeguard against data loss because of theft or loss of the Windows CE device, or media failure on the Windows CE device.

Most backup and recovery utilities are available on Windows CE. However, these utilities are not useful since you cannot use the utilities on Windows CE to store backups in a physically separate location. Instead, data can be backed up by copying the entire database file to a computer. You can also use synchronization to maintain an up-to-date copy of your Windows CE database on a computer. See "Synchronization Basics" [*MobiLink - Getting Started*].

Erasing a Windows CE database

SQL Anywhere for Windows CE does not support the Erase Database wizard, the DROP DATABASE statement, or the Erase utility (dberase). You must manually erase databases from your Windows CE device. The database must not be running when you attempt to delete it.

There are two methods for erasing a database from your Windows CE device. You can erase a database through the device interface, or you can connect your device to a 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.

   A popup menu appears.

3. Tap Delete.

4. Tap Yes to confirm the deletion.

♦ To erase a database using Windows Explorer

1. Place your Windows CE device in its cradle and ensure that it connects to the computer via ActiveSync.
2. Open Windows Explorer on your computer.
3. Browse to the Windows CE directory where the database file is stored.
4. Right-click the database file and choose Delete.

   Click Yes when you are prompted to confirm the deletion.

Running the database server on Windows CE

The usual client/server arrangement has the database server running on a computer with more power and resources than the client applications. Clearly, this is not the case with Windows CE; instead, the less powerful computer is running the database server.

The network database server is supplied for Windows CE. Its file name is *dbsrv10.exe*. The network database server supports communications over TCP/IP. Because Windows CE supports the network database server, you can run administration utilities on a computer to execute tasks on your Windows CE database. For example:

- You can use Sybase Central on your computer to manage your database.
- You can use Interactive SQL on your computer to load and unload data, and perform queries.

☞ For more information, see "Using the administration utilities on Windows CE" on page 986.

The Windows CE database server does not start the TCP/IP network link unless it is explicitly requested.

☞ For more information about starting a database server on Windows CE, see "Tutorial: Running Windows CE databases from Sybase Central" on page 986.

On Windows CE, 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 CE applications. Because of this behavior, you cannot run two database servers at the same time on a Windows CE device. However, SQL Anywhere supports running multiple databases on a single database server.

Specifying server options on Windows CE

You can specify server and database 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 CE, options are specified in the Server Startup Options dialog. 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 CE.

☞ For more information about database server options, see "The Database Server" on page 117.

☞ For information about unsupported options, see "Database server option support on Windows CE" on page 996.

Using the administration utilities on Windows CE

This section describes specific considerations for using the SQL Anywhere database administration utilities with Windows CE databases.

Tutorial: Running Windows CE 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 "SQL Anywhere for Windows CE" on page 961, "Connecting to a database running on a Windows CE device" on page 971, and "Creating an ODBC data source to connect to your Windows CE device" on page 973 and before you begin the tutorial.
- Connect your Windows CE device to a computer.

Before you begin

You need to create two Windows CE databases for use in the tutorial.

♦ To create databases for your Windows CE device

1. Ensure that your Windows CE device is connected to the computer.

2. Open Sybase Central on your computer by choosing Start ▶ Programs ▶ SQL Anywhere 10 ▶ Sybase Central.

   Sybase Central appears.

3. Start the Create Database wizard by choosing Tools ▶ SQL Anywhere 10 ▶ Create Database.

   The Create Database wizard appears.

4. Name the database **Alpha.db**. Click Next.

   You can disregard the directory that the database is stored in on the computer because the computer copy will be deleted once the database is copied to your Windows CE device.

5. Select the Create This Database for Windows CE option. Click Next.

   The wizard tests the connection to your Windows CE device.

6. Select the Copy the Database to Your Windows CE Device option.

   Copy the database to the My Documents directory on your Windows CE device by typing the following in the Windows CE File Name field: *\My Documents\Alpha.db*. Click Next.

7. Select the Delete the Desktop Database After Copying option.

8. Follow the remaining instructions in the wizard.

9. Click Finish to create the database with the default settings.

   Once the wizard finishes copying the *Alpha.db* database file to your Windows CE device, continue to the next step.

10. Repeat Steps 3 through 9 to create a database called *Beta.db*.

    The path in Step 6 for the second database is *\My Documents\Beta.db*.

Once the wizard finishes copying the *Beta.db* database file to your Windows CE device, you are ready to begin the tutorial.

Lesson 1: Starting the database server

This section describes the simple case of running a single database on Windows CE.

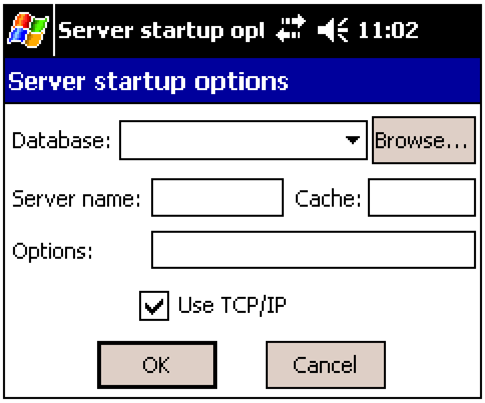
♦ To start a database on the server

1. Open File Explorer on your Windows CE device by tapping Start ► Programs ► File Explorer.

2. Navigate to the SQL Anywhere directory by tapping Program Files ► SQLAny10.

3. Tap the *dbsrv10* icon.

   The Server Startup Options dialog appears.



☑ Use TCP/IP

4. Specify the database file that you want to start.

   Tap Browse and locate the *Alpha.db* file in the *My Documents* directory.

5. Assign a name to the network database server by typing **CEserver** in the Server Name field.

6. Use the default cache size of 600 KB.

Tip
For the purposes of this tutorial, the default cache size is sufficient. However, a larger cache size can help improve performance for larger databases.
For more information, see "Use the cache to improve performance" [*SQL Anywhere Server - SQL Usage*].

7. Select the Use TCP/IP option.

   A TCP/IP connection is necessary to connect from a computer to the database running on your Windows CE device. You will connect from your computer in a later lesson.

8. In the Options field of the Server Startup Options dialog, type **-gd all**.

   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.

   ☞ For more information, see "-gd server option" on page 146.

9. Tap OK to start the Alpha database running on the network database server.

10. Navigate to the Today screen on your device.

11. Tap the database server icon located in the bottom right corner of the screen.

    The Server Messages window appears.

When the message `Now accepting requests` appears in the Server Messages window, you are ready to proceed to the next lesson.

What's next?

Next, you will learn how to start multiple databases on the network database server on Windows CE.

Lesson 2: Starting multiple databases on the Windows CE database server

On Windows CE 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 CE applications. Because of this behavior, two database servers cannot be run at the same time on a Windows CE device. As an alternative to running multiple database servers, one server can run multiple databases.

♦ To connect to a database from Sybase Central

1. Start Sybase Central on your computer by choosing Start ► Programs ► SQL Anywhere 10 ► Sybase Central.

   Sybase Central starts.

2. From the Connections menu, choose Connect with SQL Anywhere 10.

   The Connect dialog appears.

3. On the Identification tab, enter the following values:

   - **User ID** **DBA**
   - **Password** **sql**

4. Select the ODBC Data Source Name option.
5. Click Browse, choose the **CEdevice** data source that you created in "Creating an ODBC data source to connect to your Windows CE device" on page 973.
6. On the Database tab, type the server name **CEserver** in the Server Name field.
7. Click OK to connect to the *Alpha.db* database running on your Windows CE device.

Now that you have started the database server and connected to the Alpha database, you can start additional databases on your Windows CE device.

♦ To start a second database on the network database server

1. From the Context dropdown list, choose **CEserver**.
2. From the File menu, choose Start Database.

   The Start Database dialog appears.

3. In the Database File field, type the following path: **\My Documents\Beta.db**
4. Type the server name **CEserver** in the Server Name field.
5. 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 computer.

♦ To connect to the second database

1. In the right pane of Sybase Central, select the **Beta** database icon.

   This icon represents the database that you just started on your Windows CE device.

2. From the File menu, choose Connect.

   The Connect dialog appears.

3. On the Identification tab, enter the following values:

   - **User ID** **DBA**
   - **Password** **sql**

4. On the Database tab, type the database name **\My Documents\Beta.db** in the Database File field.
5. Click OK to connect to the Beta database running on your Windows CE 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 CE.

Lesson 3: Shutting down the database server on Windows CE

Before you can shut down the network database server on your Windows CE device, you must stop the connections from your computer.

♦ To disconnect from the Windows CE databases

1. In Sybase Central, choose Connections ▶ Disconnect.

   The Disconnect dialog appears.

2. Select the connection that corresponds to the Alpha database.
3. Click Disconnect.
4. Repeat Steps 1 through 3 for the connection that corresponds to the Beta database.

Now that you have disconnected from the Windows CE databases in Sybase Central, you can shut down the network database server.

♦ To shut down the server

1. Tap the database server icon located in the bottom right corner of the Today screen.

   The Server Messages window appears.

2. Tap 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 plug-in" on page 535
♦ "Working with databases" [*SQL Anywhere Server - SQL Usage*]

Tutorial: Managing Windows CE 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, as well as 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 computer to manage databases on your Windows CE device. You will learn how to connect to the sample database on your Windows CE 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 CE device before you can connect to it from Interactive SQL.

♦ To start the sample database

1. Open File Explorer on your Windows CE device by tapping Start ▶ Programs ▶ File Explorer.
2. Navigate to the SQL Anywhere directory by tapping Program Files ▶ SQLAny10.
3. Tap the dbsrv10 icon.

   The Server Startup Options dialog appears.
4. In the Database field, type the path of the sample database: **\Program Files\SQLAny10\demo.db**.
5. Assign a name to the network database server by typing **CEserver** in the Server Name field.
6. Set the cache size by typing **5MB** in the Cache field.

   The default cache size on Windows CE is 600 KB. However, a larger cache size is recommended to help improve performance.
7. Select Use TCP/IP.
8. Click OK to start the sample database running on the network database server.

   The Server Messages window appears briefly, then disappears.
9. Navigate to the Today screen on your device.
10. Tap the Server icon located in the bottom right corner of the screen.

    The Server Messages window appears.

    When the message `Now accepting requests` appears in the Server Messages window, you are ready to move on to the next lesson.

What's next?

Next you will learn how to connect from Interactive SQL to the database running on your Windows CE device.

Lesson 2: Start Interactive SQL and connect

Now that the sample database is running on your Windows CE device, connect to it from Interactive SQL to view and manage the database from your computer.

♦ To connect from Interactive SQL to a database on your Windows CE device

1. Start Interactive SQL on your computer by choosing Start ▶ Programs ▶ SQL Anywhere 10 ▶ Interactive SQL.

   The Connect dialog appears automatically when Interactive SQL starts.

2. On the Identification tab, enter the following values:

   - ♦ **User ID** **DBA**
   - ♦ **Password** **sql**

3. Select the ODBC Data Source Name option.

4. Click Browse, choose the **CEdevice** data source that you created in "Creating an ODBC data source to connect to your Windows CE device" on page 973, and then click OK.

5. On the Database tab, specify the server name in the Server Name field.

   In this example, the server name is CEserver, which you specified in the previous lesson.

6. Click OK to connect to the sample database running on your Windows CE device.

What's next?

You can now view and manage the data in the sample database from Interactive SQL.

Lesson 3: Executing queries against a Windows CE 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 CE database

1. Type the following in the SQL Statements pane:

   ```
   SELECT * FROM Employees
   ```

2. Click Execute SQL Statement, or choose SQL ▶ Execute, or press F5 to execute the statement.

   All the data from the Employees table appears in the Results pane.

3. Choose SQL ▶ Disconnect or press F12 to disconnect from the Windows CE database.

Where do I go from here?

Once you connect to a database from Interactive SQL, you can view and manipulate the dat, as well as add and modify database objects.

☞ For information about Interactive SQL, writing queries, and using SQL statements, see:

- ♦ "Interactive SQL" on page 539

- “Queries: Selecting Data from a Table” [*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 CE

This section lists the components and features of SQL Anywhere that are unsupported or have altered functionality on Windows CE. Where available, alternatives to unsupported features are listed.

☞ For more information about supported and unsupported components on Windows CE, see http://www.ianywhere.com/products/supported_platforms.html.

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 CE. Instead, database administration is performed from a Windows-based computer that is connected to the Windows CE device.

☞ For more information, see "Using the administration utilities on Windows CE" on page 986.

Component or feature	**Considerations**
Application profiling	When you create a tracing session for a database running on Windows CE, 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 CE device to a copy of the Windows CE database running on a database server on a desktop computer. You cannot automatically create a tracing database from a Windows CE device, and you cannot trace to the local database on a CE device. See "Application profiling" [*SQL Anywhere Server - SQL Usage*].
Database mirroring	Unsupported on Windows CE.
External stored procedures	Unsupported on Windows CE.
iAnywhere JDBC driver	Unsupported on Windows CE. You can use jConnect on Windows CE.
Java in the database	Unsupported on Windows CE.
jConnect	The jConnect driver can be enabled when you create a database for Windows CE. This can be useful if you want to move the database to a 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 CE.
Kerberos authentication	Unsupported on Windows CE.
LDAP authentication	Unsupported on Windows CE.

Component or feature	Considerations
ODBC clients	Windows CE 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 CE” on page 68.
Open Client	Unsupported on Windows CE.
Parallel backups	Unsupported on Windows CE.
Personal database server (dbeng10)	Only the network database server (*dbsrv10*) is supported on Windows CE. This executable supports local connections and client/server communications across a network.
Remote data access (including directory access servers)	Unsupported on Windows CE.
SPX protocol	Windows CE uses the TCP/IP protocol.

SQL statement support on Windows CE

This section describes SQL statements that are not supported on Windows CE, as well as 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 CE.
CREATE DATABASE statement	The CREATE DATABASE statement can be used to initialize a database on a computer, which can later be copied to a Windows CE device. See “Creating a Windows CE database” on page 976.
CREATE EVENT statement	DiskSpace event types are not supported on Windows CE. However, you can use this statement to define the GlobalAutoIncrement event type or the ServerIdle event type.
CREATE EXISTING TABLE statement	Unsupported on Windows CE.
CREATE EXTERNLOGIN statement	Unsupported on Windows CE.
CREATE FUNCTION statement	The CREATE FUNCTION statement can be used on Windows CE to create user-defined SQL functions for use in the database. Note that the EXTERNAL NAME clause is not supported on Windows CE.
CREATE SERVER statement	Unsupported on Windows CE.

SQL statement	Considerations
CREATE TABLE statement	The AT clause of the CREATE TABLE statement for creating proxy tables is not supported on Windows CE.
DROP DATABASE statement	Unsupported on Windows CE.
DROP SERVER statement	Unsupported on Windows CE.
INSTALL JAVA statement	Unsupported on Windows CE.
REMOVE JAVA statement	Unsupported on Windows CE.
REORGANIZE TABLE statement	Unsupported on Windows CE.
RESTORE DATABASE statement	Unsupported on Windows CE.
START JAVA statement	Unsupported on Windows CE
STOP JAVA statement	Unsupported on Windows CE.

Database server option support on Windows CE

This section describes those database server options that are not supported or have altered functionality on Windows CE.

Option	Considerations
@ data option	Unsupported on Windows CE.
-? server option	Unsupported on Windows CE.
-cm server option	Unsupported on Windows CE.
-cw server option	Unsupported on Windows CE.
-ec server option	Strong communication encryption is not supported on Windows CE. Only the **none** and **simple** settings are supported. See “-ec server option” on page 139.
-gb server option	Unsupported on Windows CE.
-ge server option	Unsupported on Windows CE.
-gss server option	Unsupported on Windows CE.
-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 CE device. This feature cannot be disabled.
-s server option	Unsupported on Windows CE.
-tmf server option	Unsupported on Windows CE.
-tmt server option	Unsupported on Windows CE.

Option	Considerations
-u server option	Unsupported on Windows CE.
-ua server option	Unsupported on Windows CE.
-uc server option	Unsupported on Windows CE.
-ud server option	Unsupported on Windows CE.
-uf server option	Unsupported on Windows CE.
-ui server option	Unsupported on Windows CE.
-ut server option	Unsupported on Windows CE.
-ux server option	Unsupported on Windows CE.
-xp server option	Unsupported on Windows CE.

Sybase Central wizard support on Windows CE

The following table lists the Sybase Central wizards that are not supported or have altered functionality on Windows CE and provides alternatives where possible.

Wizard	Considerations
Backup Database wizard	Archive backups are not supported on Windows CE. The Backup Database wizard is not supported. See “Types of backup” on page 769. As an alternative on Windows CE, you can use the Create Backup Images wizard, which makes a separate backup of the database and transaction log files. See “Create Backup Images wizard” on page 590.
Change Log File Settings wizard	Unsupported on Windows CE.
Create Database wizard	This wizard provides options for creating database on Windows CE, provided Windows CE services are installed on the computer running Sybase Central. See “Creating a Windows CE database” on page 976.
Create Maintenance Plan wizard	The following options are not available on Windows CE: ♦ Full Archive Backup ♦ Back up to Tape ♦ Email the Maintenance Plan Report
Erase Database wizard	Unsupported on Windows CE.
Migrate Database wizard	Unsupported on Windows CE.
Restore Database wizard	Unsupported on Windows CE.

Wizard	Considerations
Create Service wizard	Unsupported on Windows CE.
Translate Log File wizard	Unsupported on Windows CE.
Unload Database wizard	This wizard cannot map to the Windows CE directory where the database files are stored. However, you can unload a Windows CE database by copying it to your computer and using the Unload Database wizard.
Upgrade Database wizard	This wizard is not supported on Windows CE. However, you can upgrade a Windows CE database by copying it to your computer and using this wizard before copying the database back to your Windows CE device. See "Upgrade Database wizard" on page 701.

SQL Remote support on Windows CE

SQL Remote is supported on Windows CE with the following exceptions:

Component or feature	Considerations
Extraction utility (dbxtract)	Windows CE does not support this utility. If necessary, a Windows CE database can be copied to a computer so that the Extraction utility can be used.
MAPI message type	This message system is not supported on Windows CE. ☞ For information about supported SQL Remote message types for Windows CE, see "Supported operating systems" [*SQL Remote*].
VIM message type	This message system is not supported on Windows CE. ☞ For information about supported SQL Remote message types for Windows CE, see "Supported operating systems" [*SQL Remote*].

Index