SYBASE®

Reference: Building Blocks, Tables, and Procedures

Sybase IQ

15.1

DOCUMENT ID: DC38151-01-1510-01

LAST REVISED: July 2009

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Subject

This book provides reference material for many aspects of Sybase IQ, including SQL statements, language elements, data types, functions, system procedures, and system tables. Other books provide more context on how to perform particular tasks. This reference book is the place to look for information such as available SQL syntax, parameters, and options. For command line utility startup parameters, see the *Utility Guide*.

Audience

This manual is a reference for all users of Sybase IQ.

Related documents

The Sybase IQ 15.1 documentation set includes:

- *Release Bulletin* provides information about last-minute changes to the product and documentation.
- *Installation and Configuration Guide* provides platform-specific instructions on installing, migrating to a new version, and configuring Sybase IQ for a particular platform.
- *Advanced Security in Sybase IQ* covers the use of user encrypted columns within the Sybase IQ data repository. You need a separate license to install this product option.
- *Error Messages* lists Sybase IQ error messages referenced by Sybase error code, SQLCode, and SQLState, and SQL preprocessor errors and warnings.
- *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library* contains a concise description of the IMSL C Stat Library time series C functions. This book is only available to RAP - The Trading Edition™ Enterprise users.
- *Introduction to Sybase IQ* includes hands-on exercises for those unfamiliar with Sybase IQ or with the Sybase Central™ database management tool.
- *Large Objects Management in Sybase IQ* explains storage and retrieval of Binary Large Objects (BLOBs) and Character Large Objects (CLOBs) within the Sybase IQ data repository. You need a separate license to install this product option.

- *New Features in Sybase IQ 15.0* documents new features and behavior changes for version 15.0.
- *New Features Summary Sybase IQ 15.1* summarizes new features and behavior changes for the current version.
- *Performance and Tuning Guide* describes query optimization, design, and tuning issues for very large databases.
- *Quick Start* lists steps to build and query the demo database provided with Sybase IQ for validating the Sybase IQ software installation. Includes information on converting the demo database to multiplex.
- *Reference Manual* – includes two reference guides to Sybase IQ:
  - *Reference: Building Blocks, Tables, and Procedures* (this book) describes SQL, stored procedures, data types, and system tables that Sybase IQ supports.
  - *Reference: Statements and Options* describes the SQL statements and options that Sybase IQ supports.
- *System Administration Guide* – includes two volumes:
  - *System Administration Guide: Volume 1* describes startup, connections, database creation, population and indexing, versioning, collations, system backup and recovery, troubleshooting, and database repair.
  - *System Administration Guide: Volume 2* describes how to write and run procedures and batches, programming with OLAP, accessing remote data, setting up IQ as an Open Server, scheduling and event handling, programming with XML, and debugging.
- *User-Defined Functions Guide* provides information about the user-defined functions, their parameters, and possible usage scenarios.
- *Using Sybase IQ Multiplex* tells how to use multiplex capability, designed to manage large query loads across multiple nodes.
- *Utility Guide* provides Sybase IQ utility program reference material, such as available syntax, parameters, and options.

Sybase IQ and SQL Anywhere
Because Sybase IQ is an extension of SQL Anywhere®, a component of the SQL Anywhere package, Sybase IQ supports many of the same features as SQL Anywhere. The Sybase IQ documentation set refers you to SQL Anywhere documentation where appropriate.

Documentation for SQL Anywhere includes:

- *SQL Anywhere Server – Database Administration* describes how to run, manage, and configure SQL Anywhere databases. It describes database connections, the database server, database files, backup procedures, security, high availability, and replication with Replication Server®, as well as administration utilities and options.
- *SQL Anywhere Server – Programming* describes how to build and deploy database applications using the C, C++, Java, PHP, Perl, Python, and .NET programming languages such as Visual Basic and Visual C#. This book also describes a variety of programming interfaces such as ADO.NET and ODBC.
- *SQL Anywhere Server – SQL Reference* provides reference information for system procedures, and the catalog (system tables and views). It also provides an explanation of the SQL Anywhere implementation of the SQL language (search conditions, syntax, data types, and functions).
- *SQL Anywhere Server – SQL Usage* 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.

You can also refer to the SQL Anywhere documentation in the SQL Anywhere 11.0.1 collection at Product Manuals at http://www.sybase.com/support/manuals/ and in DocCommentXchange at http://dcx.sybase.com/dcx_home.php.

Documentation for Sybase Software Asset Management (SySAM) includes:

- *Sybase Software Asset Management (SySAM) 2* introduces asset management concepts and provides instructions for establishing and administering SySAM 2 licenses.
- *SySAM 2 Quick Start Guide* tells you how to get your SySAM-enabled Sybase product up and running.
- *FLEXnet Licensing End User Guide* explains FLEXnet Licensing for administrators and end users and describes how to use the tools that are part of the standard FLEXnet Licensing distribution kit from Sybase.

Other sources of information

Use the Sybase Getting Started CD, the SyBooks™ CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://sybooks.sybase.com.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ Finding the latest information on product certifications

1. Point your Web browser to Technical Documents at http://certification.sybase.com/ucr/search.do.
2. Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.
3. Select Search to display the availability and certification report for the selection.

❖ Finding the latest information on component certifications

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.

2 Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.

3 Select Search to display the availability and certification report for the selection.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Syntax conventions

This documentation uses these conventions in syntax descriptions:

- **Keywords** SQL keywords are shown in UPPERCASE. However, SQL keywords are case-insensitive, so you can enter keywords in any case; SELECT, Select, and select are equivalent.

- **Placeholders** Items that must be replaced with appropriate identifiers or expressions are shown in *italics*.
- **Continuation** Lines beginning with an ellipsis (...) are a continuation of the statements from the previous line.
- **Repeating items** Lists of repeating items are shown with an element of the list followed by an ellipsis (...). One or more list elements are allowed. If multiple elements are specified, they must be separated by commas.
- **Optional portions** Optional portions of a statement are enclosed by square brackets. For example:

  ```
  RELEASE SAVEPOINT [ savepoint-name ]
  ```

  The square brackets indicate that the *savepoint-name* is optional. Do not type the brackets.
- **Options** When none or only one of a list of items must be chosen, the items are separated by vertical bars and the list enclosed in square brackets. For example:

  ```
  [ ASC | DESC ]
  ```

  The square brackets indicate that you can choose ASC, DESC, or neither. Do not type the brackets.
- **Alternatives** When precisely one of the options must be chosen, the alternatives are enclosed in curly braces. For example:

  ```
  QUOTES { ON | OFF }
  ```

  The curly braces indicate that you must include either ON or OFF. Do not type the brackets.

Typographic conventions

Table 1 lists the typographic conventions used in this documentation.

Table 1: Typographic conventions

Item	Description
`Code`	SQL and program code appears in a monospaced (fixed-width) font.
`User entry`	Text entered by the user is shown in a monospaced (fixed-width) font.
emphasis	Emphasized words are shown in italic.
file names	File names are shown in italic.
database objects	Names of database objects, such as tables and procedures, are shown in bold, sans serif type in print, and in italic online.

The demo database

Sybase IQ includes scripts to create a demo database (*iqdemo.db*). Many of the queries and code samples in this document use the demo database as a data source.

The demo database contains internal information about a small company (employees, departments, and financial data), as well as product (products), and sales information (sales orders, customers, and contacts).

See the Sybase IQ installation guide for your platform or talk to your system administrator for more information about the demo database.

Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Sybase IQ 15.1 and the HTML documentation have been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

Configuring your accessibility tool

You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool for information on using screen readers.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

For a Section 508 compliance statement for Sybase IQ, go to Sybase Accessibility at http://www.sybase.com/products/accessibility.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

File Locations and Installation Settings

About this chapter

This chapter describes the installation and operating system settings used by Sybase IQ. Depending on the operating system, these settings may be stored as environment variables, initialization file entries, or registry entries.

Contents

Installation directory structure

When you install Sybase IQ, several directories may be created. The directories created depend on which options you choose during installation and which directories already exist in your Sybase directory (the directory defined by *$SYBASE* on UNIX or *%SYBASE%* on Windows). This section describes the directory structure.

By default, Sybase IQ software is installed in a unique subdirectory under the Sybase directory. This subdirectory is called the installation directory. Other tools provided with Sybase IQ have unique subdirectories under the Sybase directory. This section describes only the subdirectory structure for Sybase IQ.

The Sybase IQ directory

By default, the Sybase IQ directory is *IQ-15_1*. The location of *IQ-15_1* varies, depending on where you install Sybase IQ. The *IQ-15_1* directory is also referred to by the environment variable *$IQDIR15* on UNIX or *%IQDIR15%* on Windows.

The Sybase IQ directory holds a number of directories and files:

- *Demo* directory (*%ALLUSERSPROFILE%/SybaseIQ/demo*) – holds the tools required to build the iqdemo database. The iqdemo database files are iqdemo.db, iqdemo.iq, iqdemo.iqmsg, and iqdemo.iqtmp. The demo database itself is not shipped with Sybase IQ.

  The subdirectory */demo/adata* holds 15.x data to allow the creation of the 15.x iqdemo database. The subdirectory */demo/demodata* holds Sybase IQ 12.7 data to allow the creation of an iqdemo database that has the same schema layout and data as the IQ 12.7 asiqdemo database. Use */demo/mkiqdemo.bat* on Windows and *demo/mkiqdemo.sh* on Unix to create the 15.x iqdemo database. The iqdemo database can be used to demonstrate problems to Technical Support.

- *Scripts* directory (*IQ-15_1/scripts*) – holds some scripts used in examples and when creating catalog objects like stored procedures. *Do not edit these scripts*. If you edit, delete, or move these scripts, the server will not operate properly.

- *Samples* directories The *samples* directory contains SQL samples and user-defined function (UDF) samples. *%ALLUSERSPROFILE%/SybaseIQ/samples/sqlanywhere* contains 24 directories of SQL samples. The *sqlanywhere/c directory* holds C++ examples that illustrate using ESQL (embedded SQL) and C with SQL Anywhere. Because SQL Anywhere and Sybase IQ share common code, you can modify these examples for use with Sybase IQ. The *%ALLUSERSPROFILES%/SybaseIQ/samples/udf* directory holds sample C++ scalar and aggregate UDFs.
- *Executable* directories – hold executables, libraries, help files, and so on.

  On UNIX, executable subdirectories include *IQ-15_1* subdirectories */bin64*, */lib64*, */logfiles*, */res*, and */tix*. On Windows, these include *IQ-15_1* subdirectories *\h*, *\install*, *\java*, and *\bin32*.

How Sybase IQ locates files

To start and run, Sybase IQ must find and access several types of files. Understanding how Sybase IQ finds these files is important, to ensure that the correct files are used. Several directories or files with identical names may reside on a system. Sybase IQ uses both Adaptive Server™ Enterprise and SQL Anywhere libraries. If either of these products have already been installed on your system, note the directory where they are installed to avoid confusion.

The types of files include but are not limited to:

- Libraries – might include product libraries or system libraries. File name extensions include *.so.nnn* or *.so* on UNIX, or *.dll* or *.lib* on Windows. These files are required to run Sybase IQ. If an incorrect DLL is found, a version mismatch error may occur. For example, library files might be found in *$IQDIR15/lib64* or *$SYBASE/$SYBASE_OCS/lib64* on UNIX, or *%IQDIR15%\bin32* or *%SYBASE\%SYBASE_OCS\dll* on Windows. An empty directory, *$IQDIR15/usrlib*, lets you supersede default libraries with custom libraries and patches, because start_iq includes *usrlib* before regular library directories.
- Interface files – required to run Sybase IQ. For example, *.odbc.ini* and *utility_db.ini* on UNIX, and *util_db.ini* on Windows. For more information about these files, see the *System Administration Guide: Volume 1* and the *Installation and Configuration Guide*.

- Configuration files – used to specify connection parameters. Examples include *default.cfg* on Windows or *iqdemo.cfg*.
- Database files – store the data and metadata. For example: *iqdemo.db*, *iqdemo.iq*, *iqdemo.iqmsg*, *iqdemo.iqtmp*.
- Log files – store information about the current session on the server and connected database. For example, a server log might be named *%ALLUSERSPROFILE%/SybaseIQ/IQ15_1/logfiles/yourservername.0006.srvlog*. The database log (for example, *%ALLUSERSPROFILE%/SybaseIQ/IQ-15_1/demo/iqdemo.log*) is created when you connect to the database. For more information about these files, see the *Installation and Configuration Guide*.
- Product scripts – are sample files that show how to create, populate, and upgrade databases.
- User files – include flat files used with the LOAD command and SQL scripts used with tools such as Interactive SQL.
- Temporary files – created by Sybase IQ to store temporary information for operations like performing sorts for queries.

Some file names are specified in SQL statements and must be located at runtime. Examples of SQL statements that use file names include the following:

- INSTALL statement – the name of the file that holds Java classes.
- LOAD TABLE statement – the name of the file from which data should be loaded.
- CREATE DATABASE statement – A file name is needed for this statement and similar statements that can create files.

In some cases, Sybase IQ uses a simple algorithm to locate files. In other cases, a more extensive search is carried out.

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; that is, where the server was started.

Also, when a database server is started and a database file name (DBF parameter) is supplied, the path is interpreted as relative to the directory in which the server was started.

Extensive file searching

Sybase IQ programs, including the database server and administration utilities, carry out extensive searches for required files, such as DLLs or shared libraries. In these cases, Sybase IQ programs look for files in the following order:

1. The executable directory – the directory in which the program executable is held. Also, directories with the following paths relative to the program executable directory:
   - Parent of the executable directory.
   - A child of the parent directory named *scripts*.
2. 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.
3. Location registry entry – on a Windows installation, Sybase IQ adds a LOCATION registry entry. The indicated directory is searched, followed by the following:
   - A child named *scripts*
   - A child with the operating system name (*bin32*, *bin*, and so on)
4. System-specific directories – this includes directories where common operating system files are held, such as the Windows directory and the Windows\system directory on Windows.
5. CLASSPATH directories – for Java files, directories listed in the CLASSPATH environment variable are searched to locate files.
6. PATH directories – directories in the system path and the user's path are searched to locate files.
7. LIBRARY PATH directories – directories listed in the *LIBPATH* environment variable are searched for shared libraries.

Environment variables

Sybase IQ uses environment variables to store various types of information; not all variables need to be set in all circumstances.

Setting environment variables

Required environment variables are set by environment source files on UNIX and by the Sybase IQ installation on Windows.

❖ Running UNIX environment source files

Issue the following command to set all required environment variables.

1. For the Bourne/Korn shell:

```
. $SYBASE/IQ-15_1/IQ-15_1.sh
```

2. For the C shell:

```
source $SYBASE/IQ-15_1/IQ-15_1.csh;
rehash
```

On Windows platforms, the installation program automatically sets all environmental variables, so no changes are necessary. However, if you must set optional variables or change defaults, use one of the following procedures, as appropriate for your operating system.

❖ Setting environment variables on Windows

1. On your desktop, right-click My Computer and select Properties from the submenu.
2. Click the Advanced tab.
3. Click the Environment Variables button.

   The Environment Variables dialog opens.

   a. If the environment variable does not already exist, click New and type the variable name and its value in the spaces provided; then click OK.

   b. If the variable does exist, select it from the list of System Variables or User Variables, click Edit, and make any modifications in the Variable Value field. Then click OK to capture the setting.

   Note See the Microsoft Windows documentation for an explanation of user variables and system variables.

❖ Setting environment variables on UNIX

1. To check the setting for an environment variable, use:

   echo $*variable-name*

   For example, to see the setting for the $SYBASE variable:

```
% echo $SYBASE
/server1/users/test/sybase
```

2. In one of your startup files (*.cshrc*, *.shrc*, *.login*), add a line that sets the variable.

   In some shells (such as sh, bash, ksh) the line is:

   ```
   VARIABLE=value;export VARIABLE
   ```

   In other shells (such as csh, tsch) the line is:

   ```
   setenv VARIABLE "value"
   ```

For details about variables Sybase IQ uses, see:

- "IQDIR15 environment variable" on page 7
- "IQPORT environment variable" on page 8
- "IQLOGDIR15 environment variable" on page 8
- "IQTIMEOUT environment variable" on page 9
- "IQTMP15 environment variable" on page 9
- "LIBPATH or LD_LIBRARY_PATH environment variable" on page 10
- "PATH environment variable" on page 11
- "SACHARSET environment variable" on page 11
- "SALANG environment variable" on page 11
- "SQLCONNECT environment variable" on page 12
- "SYBASE environment variable" on page 12
- "$SYBASE_JRE6_32, $SYBASE_JRE6_64, $SYBASE_JRE5_64 environment variables" on page 13
- "SYBASE_OCS environment variable" on page 13

IQDIR15 environment variable

Setting

```
IQDIR15 = ${SYBASE}/IQ-15_1
```

Operating system

(Required) Set by the environment source file or the installation program. This default setting can be changed on Windows.

Description

IQDIR15 identifies the location of the Sybase IQ directory and is the location for other directories and files under that directory:

- *$IQDIR15/bin[64]/util_db.ini* holds the login ID and password for the utility database, utility_db. The installation program lets you change these from their default values, login ID "DBA" and password "sql."
- *$IQDIR15/logfiles* is the default location for the server log and backup/restore log (the backup history file). You can override this default by setting the IQLOGDIR15 environment variable.
- *$IQDIR15/demo* is the location for the iqdemo database files.

IQPORT environment variable

Setting

```
IQPORT = 5556
```

Operating system

Optional. If the user did not specify IQPORT in the environment source file, the port number defaults to 1099. You can change this default value, provided you do so before the plug-in starts. You can set this variable as described in "Setting environment variables" on page 6 or by supplying the -DIQPORT argument to the scjview command when starting Sybase Central. For example:

```
scjview -DIQPORT=3345
```

Description

Overrides the default value for the Sybase IQ Agent port number, which is used for communications between the Sybase IQ plug-in and Agent.

Note Once the agent starts, you cannot change the port value.

1099 is the plug-in default value when searching for an agent process on any given port. If the plug-in finds no agent on this port, it displays a prompt so that you can specify the correct port value.

IQLOGDIR15 environment variable

Setting

```
IQLOGDIR15 = path
```

Operating system

Optional.

Description

The IQLOGDIR15 environment variable is not set by the installation program. It defines the location of various log files:

- The server log is in the file *servername.nnnn.srvlog* (where nnnn is the number of times the server has been started) in the directory specified by $IQLOGDIR15.

If IQLOGDIR15 is not set to a valid, write-enabled directory, then most utilities, including start_iq, use the default location *$IQDIR15/logfiles* for all server logs.

IQTIMEOUT environment variable

Setting

IQTIMEOUT = *nnn*

Operating system

Optional but recommended in multiplex environments. For information on multiplex capability, see *Using Sybase IQ Multiplex*.

Description

The Sybase IQ Agent waits indefinitely for a process to complete. Setting a wait time is recommended when creating or synchronizing query servers for a multiplex with a very large catalog store. Large catalog stores extend the time needed for the dbbackup part of synchronization, and increasing the wait time accommodates a larger synchronize.

IQTIMEOUT overrides the default wait time of five minutes, and the argument *nnn* is the number of minutes for the Sybase IQ Agent to wait. For example:

- To wait 45 minutes (Korn or Bourne shell):

```
IQTIMEOUT=45
export IQTIMEOUT
```

- To wait an hour (C shell):

```
setenv IQTIMEOUT 60
```

Note Set IQTIMEOUT before you invoke the agent startup option. See "Before you Install" and "Starting the Sybase IQ Agent" in the *Installation and Configuration Guide* and "Running the Sybase IQ Agent" in *Introduction to Sybase IQ*.

IQTMP15 environment variable

Setting

`IQTMP15 = temp_directory`

Operating system

Optional on UNIX. Not used on Windows platforms.

Description

The IQTMP15 environment variable is not set by the installation program. IQTMP15 is used by Sybase IQ to indicate a directory where temporary files are kept.

The IQTMP15 environment variable should point to a local directory for those using NFS (network file system), which permits the IQTMP15 directory to purge directories and files that are no longer needed as client connections are closed. Each client connection creates several directories and files in a temporary directory. These are needed only for the duration of the connection. The directory must have write permissions for all users who connect to the server.

Note The temporary files whose location is defined by IQTMP15 are files used by the client and server. This variable does not control the default location of your IQ temporary store. For information on how Sybase IQ determines the location of your temporary store, see the CREATE DATABASE statement in Chapter 1, “SQL Statements,” in *Reference: Statements and Options*.

If you do not explicitly set IQTMP15, or if it is set to $SYBASE or $IQDIR15, then the Sybase IQ Agent sets IQTMP15 to a subdirectory in the UNIX directory */tmp*.

If more than one database server is running on a machine, each server and associated local client needs a separate temporary directory to avoid conflicts. When you do not specify a port or engine number for connection, Sybase IQ uses shared memory connectivity instead of network connectivity.

To avoid conflicts when using shared memory:

- Create a temporary directory dedicated to each server. Make sure that each local client uses the same temporary directory as its server by setting the IQTMP15 environment variable explicitly in both environments.
- Create a data source name in the *.odbc.ini* file (on UNIX) for each server and provide detailed connection information. See the *Installation and Configuration Guide*.
- Use connection strings that specify explicit parameters instead of relying on defaults.
- Confirm connections by issuing:

```
SELECT "database name is" = db_name(),
"servername_is" = @@servername
```

LIBPATH or LD_LIBRARY_PATH environment variable

Settings

```
For AIX:
```

```
LIBPATH = installation_path/lib

For all other UNIX/LINUX platforms:
LD_LIBRARY_PATH = installation_path/lib
```

Operating system — Required. Variable name is platform dependent. UNIX only.

Description — Specifies the directories where Sybase IQ shared libraries are located. On UNIX, set the library path variable by running the environment source file.

PATH environment variable

Setting

```
PATH = installation_path
```

Operating system — Required.

Description — The PATH environment variable is an operating system required variable that includes the directories where Sybase IQ executables are located. On Windows, the installation program modifies PATH. On UNIX, run the environment source file to include the necessary directories.

On Windows, PATH takes the place of the LIBRARY_PATH variable, so executables and DLLs are located using the PATH variable.

SACHARSET environment variable

Setting — **SACHARSET**=*charset*

Description — *Charset* is a character set name. For example, setting SACHARSET=cp1252 sets the default character set to cp1252.

The first of the following values set determines the default character set.

- SACHARSET environment variable
- Query the operating system

If no character set information is specified, use iso_1 for UNIX, or cp850 otherwise.

SALANG environment variable

Setting — **SALANG**=*language_code*

Operating system — Optional but recommended in non-English environments.

Description

Language_code is the two-letter combination representing a language. For example, setting SALANG=DE sets the default language to German.

The first of the following values set determines the default language.

- SALANG environment variable
- Registry (Windows only) as set by the installer or *dblang.exe*
- Query the operating system

If no language information is set, English is the default.

SQLCONNECT environment variable

Settings

```
SQLCONNECT = parameter#value ; ...
```

Operating system

Optional.

Description

The SQLCONNECT environment variable is optional, and is not set by the installation program.

SQLCONNECT specifies connection parameters that are used by several of the database administration utilities, such as DBISQL, DBINFO, DBCOLLAT, and DBSTOP, when connecting to a database server. This string is a list of parameter settings, of the form parameter=*value*, delimited by semicolons.

The number sign "#" is an alternative to the equals sign; use it if you are setting the connection parameters string in the SQLCONNECT environment variable. Using "=" inside an environment variable setting is a syntax error. The = sign is allowed only in Windows.

Note Specifying connection parameters in SQLCONNECT rather than on the command line offers greater security on UNIX systems. It prevents users from being able to display your password with the ps -ef command. This is especially useful if you run DBISQL or other utilities in quiet mode.

See also

See "Connection parameters" in Chapter 4, "Connection and Communication Parameters," in the *System Administration Guide: Volume 1*.

SYBASE environment variable

Setting

```
SYBASE = path
```

Operating system Required.

Description SYBASE identifies the location of Sybase applications, such as Open Client and Open Server. You must set the SYBASE variable before you can install Sybase IQ on UNIX systems. This variable is required for using Sybase Central on UNIX systems.

$SYBASE_JRE6_32, $SYBASE_JRE6_64, $SYBASE_JRE5_64 environment variables

Setting

```
SYBASE_JRE= "${SYBASE}/shared/jre-6_0"
```

Description This variable specifies the location of the Java Runtime Environment used by the Sybase Central plug-in for Sybase IQ. For Windows and UNIX, the environment variable is $SYBASE_JRE6_32 or $SYBASE_JRE6_64. For AIX/LinuxIBM the variable is $SYBASE_JRE5_64.

On UNIX, run the *SYBASE.csh* (C shell) or *SYBASE.sh* (Bourne or Korn shell) environment source file. On Windows, the installation program sets the variable when it installs Open Client Software Developer's Kit.

SYBASE_OCS environment variable

Setting

```
SYBASE_OCS = "OCS-15_1"
```

Operating system Required.

Description The SYBASE_OCS variable specifies the home directory for the Open Client product. This variable is only used on Window. On Windows, the installation program sets SYBASE_OCS when it installs Open Client/Server Software Developers Kit.

Registry entries

On Windows operating systems, Sybase IQ uses several Registry settings. These settings are made for you by the software, and in general operation, you should not need to access the registry. The settings are provided here if you modify your operating environment.

Warning! Sybase recommends *not* modifying the Registry, as incorrect changes might damage your system.

Current user and local machine settings

Some operating systems, such as Windows, 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 machine and are available no matter which user is logged on; these are called local machine settings. You must have administrator permissions on your machine to make local machine settings.

Sybase IQ permits the use of both current user and local machine settings. For Windows, these settings are held in the HKEY_CURRENT_USER registry and HKEY_LOCAL_MACHINE registry, respectively.

The Sybase IQ installation lets you choose whether the settings it makes are for the current user only or at the local machine level.

If you make settings in both 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 Sybase IQ program as a service on Windows, 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 machine, as long as you do not shut the machine down entirely. Services can be made independent of individual accounts and need access to local machine settings.

In general, Sybase recommends using local machine settings.

Registry structure

On Windows, you can access the Registry directly using the Registry Editor.

To start the editor, select Start > Run and type in the Open box

```
regedt32
```

Note Read Only Mode protects your Registry data from accidental changes. To use it, open the Registry Editor, select Edit >Permissions, and then check Read permission.

The Sybase IQ registry entry is held in the HKEY_LOCAL_MACHINE key, in the following location:

```
SOFTWARE
    Sybase
        IQ 15.0
```

Registry settings on installation

The installation program automatically makes the following registry settings in the Sybase registry:

- Location – In the Sybase IQ registry, this entry holds the installation directory location. For example:

  ```
  Location:REG_SZ:C:\Program Files\Sybase
  \IQ-15_1
  ```

The Sybase IQ registry includes other entries for installed applications. The Sybase Central registry holds information about the Sybase Central version and installed plug-ins.

CHAPTER 2 **SQL Language Elements**

About this chapter

This chapter presents detailed descriptions of the language elements and conventions of Sybase IQ SQL.

Contents

Keywords

Each SQL statement contains one or more keywords. SQL is not case-sensitive to keywords, but throughout these manuals, keywords are indicated in uppercase.

For example, in the following statement, SELECT and FROM are keywords:

```
SELECT *
FROM Employees
```

The following statements are equivalent to the one above:

```
Select *
From Employees
select * from Employees
sELECT * FRoM Employees
```

Reserved words

Some keywords in SQL are also **reserved words**. To use a reserved word in a SQL statement as an identifier, you must enclose the word in double quotes. Many, but not all, of the keywords that appear in SQL statements are reserved words. For example, you must use the following syntax to retrieve the contents of a table named SELECT.

```
SELECT *
FROM "SELECT"
```

Because SQL is not case-sensitive with respect to keywords, each of the words in Table 2-1 may appear in uppercase, lowercase, or any combination of the two. All strings that differ only in capitalization from these words are reserved words.

If you are using Embedded SQL, you can use the database library function sql_needs_quotes to determine whether a string requires quotation marks. A string requires quotes if it is a reserved word or if it contains a character not ordinarily allowed in an identifier.

Table 2-1 lists the SQL reserved words in Sybase IQ.

Table 2-1: SQL reserved words

active	add	all	algorithm
alter	and	any	append
as	asc	auto	backup
begin	between	bigint	binary
bit	bottom	break	by
calibrate	calibration	call	cancel
capability	cascade	case	cast
certificate	char	char_convert	character
check	checkpoint	checksum	clientport
close	columns	comment	commit
committed	comparisons	computes	conflict
connect	constraint	contains	continue
convert	create	cross	cube
current	current_timestamp	current_user	cursor
date	dbspace	dbspacename	deallocate
debug	dec	decimal	declare
decoupled	decrypted	default	delay
delete	deleting	density	desc
deterministic	disable	distinct	do
double	drop	dynamic	elements
else	elseif	enable	encapsulated
encrypted	end	endif	escape
except	exception	exclude	exec
execute	existing	exists	explicit
express	externlogin	fastfirstrow	fetch
first	float	following	for
force	foreign	forward	from
full	gb	goto	grant
group	grouping	having	hidden
history	holdlock	identified	if
in	inactive	index	index_lparen
inner	inout	input	insensitive
insert	inserting	install	instead
int	integer	integrated	intersect
into	iq	is	isolation
jdk	join	kb	key
lateral	left	like	lock

logging	login	long	mb
match	membership	message	mode
modify	namespace	natural	new
no	noholdlock	nolock	not
notify	null	numeric	of
off	on	open	optimization
option	options	or	order
others	out	outer	over
pages	paglock	partial	partition
passthrough	password	plan	preceding
precision	prepare	primary	print
privileges	proc	procedure	proxy
publication	raiserror	range	raw
readcommitted	readonly	readpast	readtext
readuncommitted	readwrite	real	recursive
reference	references	release	relocate
remote	remove	rename	reorganize
repeatable	repeatableread	reserve	resizing
resource	restore	restrict	return
revoke	right	rollback	rollup
root	row	rowlock	rows
save	savepoint	schedule	scroll
secure	select	sensitive	serializable
service	session	set	setuser
share	smallint	soapaction	some
space	sqlcode	sqlstate	start
stop	subtrans	subtransaction	synchronize
syntax_error	table	tablock	tablockx
tb	temporary	then	ties
time	timestamp	tinyint	to
top	tran	transaction	transactional
transfer	tries	trigger	truncate
tsequal	unbounded	uncommitted	union
unique	uniqueidentifier	unknown	unsigned
update	updating	updlock	url
user	utc	using	validate
values	varbinary	varchar	variable
varying	virtual	view	wait

waitfor	web	when	where
while	window	with	withauto
with_cube	with_lparen	with_rollup	within
word	work	writeserver	writetext
xlock	xml		

Identifiers

Function

Identifiers are names of objects in the database, such as user IDs, tables, and columns.

Description

Identifiers have a maximum length of 128 bytes. They must be enclosed in double quotes or square brackets if any of the following conditions are true:

- The identifier contains spaces.
- The first character of the identifier is not an alphabetic character (as defined below).
- The identifier contains a reserved word.
- The identifier contains characters other than alphabetic characters and digits.

  Alphabetic characters include the alphabet, as well as the underscore character (_), at sign (@), number sign (#), and dollar sign ($). The database collation sequence dictates which characters are considered alphabetic or digit characters.

You can represent an apostrophe (single quote) inside an identifier by following it with another apostrophe.

Identifiers have the following limitations:

- Table names cannot contain double quotes.
- User names and database names cannot contain double quote, single quote, and semi-colon characters.
- User names and database names cannot start or end with a space.
- Dbspace names are not case-sensitive in a CASE RESPECT database.

If the QUOTED_IDENTIFIER database option is set to OFF, double quotes are used to delimit SQL strings and cannot be used for identifiers. However, you can always use square brackets to delimit identifiers, regardless of the setting of QUOTED_IDENTIFIER.

The default setting for the QUOTED_IDENTIFIER option is OFF for Open Client and jConnect connections; otherwise the default is ON.

Examples

The following are all valid identifiers.

```
Surname
"Surname"
[Surname]
SomeBigName
"Client Number"
```

See also

For a complete list of reserved words, see "Reserved words" on page 18.

For information on the QUOTED_IDENTIFIER option, see "The quoted_identifier option" on page 32.

For additional restrictions on server and database names, see "Server command-line switches" on page 7 in Chapter 1, "Running the Database Server" of the *Utility Guide*.

Strings

Strings are of the following types:

- Literal strings
- Expressions with CHAR or VARCHAR data types.

An expression with a CHAR data type might be a built-in or user-defined function, or one of the many other kinds of expressions available.

For more information on expressions, see "Expressions" on page 23.

A literal string is any sequence of characters enclosed in apostrophes ('single quotes'). A SQL variable of character data type can hold a string. This is a simple example of a literal string:

```
'This is a string.'
```

Special characters in strings

Represent special characters in strings by escape sequences, as follows:

- To represent an apostrophe inside a string, use two apostrophes in a row. For example:

```
'John''s database'
```

- To represent a newline character, use a backslash followed by n (\n). For example:

```
'First line:\nSecond line:'
```

- To represent a backslash character, use two backslashes in a row (\\). For example:

```
'c:\\temp'
```

- Hexadecimal escape sequences can be used for any character, printable or not. A hexadecimal escape sequence is a backslash followed by an x followed by two hexadecimal digits (for example, \x6d represents the letter m). For example:

```
'\x00\x01\x02\x03'
```

Compatibility

For compatibility with Adaptive Server Enterprise, you can set the QUOTED_IDENTIFIER database option to OFF. With this setting, you can also use double quotes to mark the beginning and end of strings. The option is ON by default.

Expressions

Syntax

expression:
case-expression
| *constant*
| [*correlation-name*.] *column-name* [*java-ref*]
| - *expression*
| *expression operator expression*
| (*expression*)
| *function-name* (*expression*, …)
| *if-expression*
| [*java-package-name.*] *java-class-name java-ref*
| (*subquery*)
| *variable-name* [*java-ref*]

Parameters

case-expression:
{ **CASE** *search-condition*
... **WHEN** *expression* **THEN** *expression* [, …]
... [**ELSE** *expression*]

END
| **CASE**
... **WHEN** *search-condition* **THEN** *expression* [, …]
... [**ELSE** *expression*]
END }

constant:
{ *integer* | *number* | *'string'* | *special-constant* | *host-variable* }

special-constant:
{ **CURRENT** { **DATE** | **TIME** | **TIMESTAMP** | **USER** }
| **LAST USER**
| **NULL**
| **SQLCODE**
| **SQLSTATE** }

if-expression:
IF *condition*
... **THEN** *expression*
... [**ELSE** *expression*]
ENDIF

java-ref:
{ **.***field-name* [*java-ref*]
| >> *field-name* [*java-ref*]
| **.***method-name* ([*expression*] [, …]) [*java-ref*]
| >> *method-name* ([*expression*] [, …]) [*java-ref*] }

operator:
{ **+** | **-** | ***** | **/** | **||** | **%** }

Usage

Anywhere

Authorization

Must be connected to the database

Side effects

None

Description

Expressions are formed from several different kinds of element, discussed in the following sections.

Compatibility

- The IF condition is not supported in Adaptive Server Enterprise.
- Java expressions are not currently supported in Adaptive Server Enterprise.
- For other differences, see the separate descriptions of each class of expression, in the following sections.

Constants in expressions

Constants are numbers or strings. String constants are enclosed in apostrophes. An apostrophe is represented inside the string by two apostrophes in a row.

Column names in expressions

A column name is an identifier preceded by an optional correlation name. A correlation name is usually a table name. For more information on correlation names, see FROM clause in *Reference: Statements and Options*. If a column name has characters other than letters, digits, and underscores, the name must be surrounded by quotation marks (“”). For example, the following are valid column names:

```
Employees.Surname
City
"StartDate"
```

See “Identifiers” on page 21.

Subqueries in expressions

A subquery is a SELECT statement enclosed in parentheses. The SELECT statement can contain one and only one select list item. When used as an expression, a scalar subquery is allowed to return only zero or one value;

Within the SELECT list of the top level SELECT, or in the SET clause of an UPDATE statement, you can use a scalar subquery anywhere that you can use a column name. However, the subquery cannot appear inside a conditional expression (CASE, IF, NULLIF, ARGN).

For example, the following statement returns the number of employees in each department, grouped by department name:

```
SELECT DepartmentName, COUNT(*), ‘out of’,
(SELECT COUNT(*) FROM Employees)
FROM Departments AS D, Employees AS E
WHERE D.DepartmentID = E.DepartmentID
GROUP BY DepartmentName;
```

For other uses of subqueries, see “Subqueries in search conditions” on page 35.

SQL operators

This section describes the arithmetic, string, and bitwise operators available in Sybase IQ. For information on comparison operators, see “Search conditions” on page 33.

The normal precedence of operations applies. Expressions in parentheses are evaluated first; then multiplication and division before addition and subtraction. String concatenation occurs after addition and subtraction.

Arithmetic operators

expression + expression Addition. If either expression is the NULL value, the result is the NULL value.

expression - expression Subtraction. If either expression is the NULL value, the result is the NULL value.

- expression Negation. If the expression is the NULL value, the result is the NULL value.

expression * expression Multiplication. If either expression is the NULL value, the result is the NULL value.

expression / expression Division. If either expression is the NULL value or if the second expression is 0, the result is the NULL value.

expression % expression Modulo finds the integer remainder after a division involving two whole numbers. For example, 21 % 11 = 10 because 21 divided by 11 equals 1 with a remainder of 10.

String operators

expression || expression String concatenation (two vertical bars). If either string is the NULL value, the string is treated as the empty string for concatenation.

expression + expression Alternative string concatenation. When using the + concatenation operator, you must ensure the operands are explicitly set to character data types rather than relying on implicit data conversion.

The result data type of a string concatenation operator is a LONG VARCHAR. If you use string concatenation operators in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set LEFT to the correct data type and size.

See “REVERSE function [String]” on page 244 for information and usage.

Standards and compatibility

- **SQL92** The || operator is the SQL92 string concatenation operator.
- **Sybase** The + operator is supported by Adaptive Server Enterprise.

Bitwise operators

You can use the following operators on all unscaled integer data types, in both Sybase IQ and Adaptive Server Enterprise.

Operator	Description
&	AND
|	OR
^	EXCLUSIVE OR
~	NOT

The AND operator (&)

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
0	1	0
1	0	0
1	1	1

The AND operator compares 2 bits. If they are both 1, the result is 1.

Bitwise OR (|)

Bit 1	Bit 2	Bit 1 | Bit 2
0	0	0
0	1	1
1	0	1
1	1	1

The OR operator compares 2 bits. If one or the other bit is 1, the result is 1.

EXCLUSIVE OR (^)

The EXCLUSIVE OR operator results in a 1 when either, but not both, of its two operands is 1.

Bit 1	Bit 2	Bit 1 ^Bit 2
0	0	0
0	1	1

Bit 1	Bit 2	Bit 1 ^Bit 2
1	0	1
1	1	0

NOT (~)

The NOT operator is a unary operator that returns the inverse of its operand.

Bit	~ Bit
1	0
0	1

Join operators

The Transact-SQL™ outer join operators *= and =* are supported in Sybase IQ, in addition to the SQL92 join syntax using a table expression in the FROM clause.

Compatibility

- **Modulo** The default value is OFF for new databases.
- **String concatenation** When you are using the + concatenation operator in Sybase IQ, ensure the operands are explicitly set to strings rather than relying on implicit data conversion. For example, the following query returns the integer value 579:

```
SELECT 123 + 456
```

  whereas the following query returns the character string 123456:

```
SELECT '123' + '456'
```

  You can use the CAST or CONVERT function to explicitly convert data types.

  Note When used with BINARY or VARBINARY data types, the + operator is concatenation, not addition.

  The || concatenation operator is not supported by Adaptive Server Enterprise.

Operator precedence

When you are using more than one operator in an expression, Sybase recommends that you use parentheses to make the order of operation explicit, rather than relying on an identical operator precedence between Adaptive Server Enterprise and Sybase IQ.

IF expressions

The syntax of the IF expression is as follows:

IF *condition*
THEN *expression1*
[**ELSE** *expression2*]
ENDIF

This expression returns:

- If *condition* is TRUE, the IF expression returns *expression1*.
- If *condition* is FALSE, the IF expression returns *expression2*.
- If *condition* is FALSE, and there is no *expression2,* the IF expression returns NULL.
- If condition is NULL, the IF expression returns NULL.

For more information about TRUE, FALSE, and UNKNOWN conditions, see "NULL value" on page 67 and "Search conditions" on page 33.

IF statement is different from IF expression
Do not confuse the syntax of the IF expression with that of the IF statement.

See IF statement in *Reference: Statements and Options*.

CASE expressions

The CASE expression provides conditional SQL expressions. You can use case expressions anywhere you can use an expression.

The syntax of the CASE expression is as follows:

CASE *expression*
WHEN *expression* **THEN** *expression* [, …]
[**ELSE** *expression*] **END**

You cannot use a subquery as a value expression in a CASE statement.

If the expression following the CASE statement is equal to the expression following the WHEN statement, then the expression following the THEN statement is returned. Otherwise, the expression following the ELSE statement is returned, if it exists.

For example, the following code uses a case expression as the second clause in a SELECT statement.

```
SELECT ID,
  (CASE name
  WHEN 'Tee Shirt' THEN 'Shirt'
  WHEN 'Sweatshirt' THEN 'Shirt'
  WHEN 'Baseball Cap' THEN 'Hat'
  ELSE 'Unknown'
  END) as Type
FROM "GROUPO".Products
```

An alternative syntax is:

CASE
WHEN *search-condition* **THEN** *expression* [, …]
[**ELSE** *expression*] **END**

If the search condition following the WHEN statement is satisfied, the expression following the THEN statement is returned. Otherwise the expression following the ELSE statement is returned, if it exists.

For example, the following statement uses a case expression as the third clause of a SELECT statement to associate a string with a search condition.

```
SELECT ID, name,
  (CASE
  WHEN name='Tee Shirt' THEN 'Sale'
  WHEN quantity >= 50  THEN 'Big Sale'
  ELSE 'Regular price'
  END) as Type
FROM "GROUPO".Products
```

NULLIF function for abbreviated CASE expressions

The NULLIF function provides a way to write some CASE statements in short form. The syntax for NULLIF is as follows:

NULLIF (*expression-1*, *expression-2*)

NULLIF compares the values of the two expressions. If the first expression equals the second expression, NULLIF returns NULL. If the first expression does not equal the second expression, NULLIF returns the first expression.

Compatibility of expressions

Table 2-2 and Table 2-3 describe the compatibility of expressions and constants between Adaptive Server Enterprise (ASE) and Sybase IQ. These tables are a guide only, and a marking of Both may not mean that the expression performs in an identical manner for all purposes under all circumstances. For detailed descriptions, see the Adaptive Server Enterprise documentation and the Sybase IQ documentation on the individual expression.

In Table 2-2, expr represents an expression, and op represents an operator.

Table 2-2: Compatibility of expressions between ASE and Sybase IQ

Expression	Supported by
constant	Both
column name	Both
variable name	Both
function (expr)	Both
- expr	Both
expr op expr	Both
(expr)	Both
(subquery)	Both
if-expression	Sybase IQ only

Table 2-3: Compatibility of constants between ASE and Sybase IQ

Constant	Supported by
integer	Both
number	Both
'string'	Both
special-constant	Both
host-variable	Sybase IQ

Default interpretation of delimited strings

By default, Adaptive Server Enterprise and Sybase IQ give different meanings to delimited strings: that is, strings enclosed in apostrophes (single quotes) and in quotation marks (double quotes).

Sybase IQ employs the SQL92 convention, that strings enclosed in apostrophes are constant expressions, and strings enclosed in quotation marks (double quotes) are delimited identifiers (names for database objects). Adaptive Server Enterprise employs the convention that strings enclosed in quotation marks are constants, whereas delimited identifiers are not allowed by default and are treated as strings.

The quoted_identifier option

Both Adaptive Server Enterprise and Sybase IQ provide a quoted_identifier option that allows the interpretation of delimited strings to be changed. By default, the quoted_identifier option is set to OFF in Adaptive Server Enterprise, and to ON in Sybase IQ.

You cannot use SQL reserved words as identifiers if the quoted_identifier option is off.

For a complete list of reserved words, see Table 2-1 on page 19.

Setting the option

Although the Transact-SQL SET statement is not supported for most Adaptive Server Enterprise connection options, SET is supported for the quoted_identifier option.

The following statement in either Sybase IQ or Adaptive Server Enterprise changes the setting of the quoted_identifier option to ON:

```
SET quoted_identifier ON
```

With the quoted_identifier option set to ON, Adaptive Server Enterprise allows table, view, and column names to be delimited by quotes. Other object names cannot be delimited in Adaptive Server Enterprise.

The following statement in Sybase IQ or Adaptive Server Enterprise changes the setting of the quoted_identifier option to OFF:

```
SET quoted_identifier OFF
```

You can choose to use either the SQL92 or the default Transact-SQL convention in both Adaptive Server Enterprise and Sybase IQ as long as the quoted_identifier option is set to the same value in each DBMS.

Examples

If you operate with the quoted_identifier option ON (the default Sybase IQ setting), the following statements involving the SQL keyword user are valid for both types of DBMS.

```
CREATE TABLE "user" (
    col1 char(5)
) ;
INSERT "user" ( col1 )
VALUES ( 'abcde' ) ;
```

If you operate with the quoted_identifier option OFF (the default Adaptive Server Enterprise setting), the following statements are valid for both types of DBMS.

```
SELECT *
FROM Employees
```

```
WHERE Surname = "Chin"
```

Search conditions

Function

To specify a search condition for a WHERE clause, a HAVING clause, a CHECK clause, a JOIN clause, or an IF expression.

Syntax

{ *expression compare expression*
| *expression compare* { **ANY** | **SOME**| **ALL** } (*subquery*)
| *expression* **IS** [**NOT**] **NULL**
| *expression* [**NOT**] **BETWEEN** *expression* **AND** *expression*
| *expression* [**NOT**] **LIKE** *expression* [**ESCAPE** *expression*]
| *expression* [**NOT**] **IN** ({ *expression* | *subquery* |
... *value-expr1* , *value-expr2* [, *value-expr3*] … })
| *column-name* [**NOT**] **CONTAINS** (… *word1* [, *word2*,] [, *word3*] …)
| **EXISTS** (*subquery*)
| **NOT** *condition*
| *condition* **AND** *condition*
| *condition* **OR** *condition*
| (*condition*)
| (*condition* , *estimate*)
| *condition* **IS** [**NOT**] { **TRUE** | **FALSE** | **UNKNOWN** }

Parameters

compare:
{ **=** | **>** | **<** | **>=** | **<=** | **<>** | **!=** | **!<** | **!>** }

Usage

Anywhere

Authorization

Must be connected to the database

Example

For example, the following query retrieves the names and birth years of the oldest employees:

```
SELECT Surname, BirthDate
FROM Employees
WHERE BirthDate <= ALL (SELECT BirthDate FROM
Employees);
```

The subqueries that provide comparison values for quantified comparison predicates might retrieve multiple rows but can have only one column.

Side effects

None

See also

“Expressions” on page 23.

Description

Conditions are used to choose a subset of the rows from a table, or in a control statement such as an IF statement to determine control of flow.

SQL conditions do not follow Boolean logic, where conditions are either true or false. In SQL, every condition evaluates as one of TRUE, FALSE, or UNKNOWN. This is called three-valued logic. The result of a comparison is UNKNOWN if either value being compared is the NULL value. For tables showing how logical operators combine in three-valued logic, see “Three-valued logic” on page 46.

Rows satisfy a search condition if and only if the result of the condition is TRUE. Rows for which the condition is UNKNOWN do not satisfy the search condition. For more information, see “NULL value” on page 67.

Subqueries form an important class of expression that is used in many search conditions. For more information, see “Subqueries in search conditions” on page 35.

The different types of search conditions are discussed in the following sections.

Comparison conditions

The syntax for comparison conditions is as follows:

expression compare expression

where *compare* is a comparison operator. Table 2-4 lists the comparison operators available in Sybase IQ.

Table 2-4: Comparison operators available in Sybase IQ

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to
<>	Not equal to
!>	Not greater than
!<	Not less than

Example

For example, the following query retrieves the names and birth years of the oldest employees:

```
SELECT Surname, BirthDate
FROM Employees
WHERE Surname <= ALL (SELECT MIN(BirthDate) FROM
```

```
Employees);
```

The subqueries that provide comparison values for quantified comparison predicates, as in the preceding example, might retrieve multiple rows but can only have one column.

Note All string comparisons are:

- Case-sensitive if the database was created as case respect (the default)
- Case-insensitive if the database was created as case ignore

See the "Usage" section of the CREATE DATABASE statement in *Reference: Statements and Options* for more details on the results of comparisons in a case-insensitive database.

Compatibility

- **Trailing blanks** Any trailing blanks in character data are ignored for comparison purposes by Adaptive Server Enterprise. The behavior of Sybase IQ when comparing strings is controlled by the Ignore Trailing Blanks in String Comparisons database creation option.
- **Case sensitivity** By default, Sybase IQ databases, like Adaptive Server Enterprise databases, are created as case-sensitive. Comparisons are carried out with the same attention to case as the database they are operating on. You can control the case sensitivity of Sybase IQ databases when creating the database.

Subqueries in search conditions

A subquery is a SELECT statement enclosed in parentheses. Such a SELECT statement must contain one and only one select list item.

A column can be compared to a subquery in a comparison condition (for example, >,<, or !=) as long as the subquery returns no more than one row. If the subquery (which must have one column) returns one row, the value of that row is compared to the expression. If a subquery returns no rows, its value is NULL.

Subqueries that return exactly one column and any number of rows can be used in IN conditions, ANY conditions, ALL conditions, or EXISTS conditions. These conditions are discussed in the following sections.

Sybase IQ supports UNION only in uncorrelated subquery predicates, not in scalar value subqueries or correlated subquery predicates.

Subqueries cannot be used inside a CONTAINS or LIKE predicate.

Sybase IQ does not support multiple subqueries in a single OR clause. For example, the following query has two subqueries joined by an OR:

```
CREATE VARIABLE @ln int;

SELECT @ln = 1;select count(*) FROM lineitem

WHERE l_shipdate IN (select l_shipdate FROM lineitem
WHERE l_orderkey IN (2,4,6))

OR l_shipdate IN (select l_shipdate FROM lineitem WHERE
l_orderkey IN (1,3,5))

OR l_linenumber = @ln;
```

Similar subqueries joined by AND and BETWEEN are allowed.

See “Comparison conditions” on page 34.

Disjunction of subquery predicates

The SQL89 standard allows for several forms of subquery predicates. Each subquery can appear within the WHERE or HAVING clause with other predicates, and can be combined using the AND or OR operators. Sybase IQ supports these subqueries, which can be correlated (contain references to a table that appears in the outer query and cannot be evaluated independently) and uncorrelated (do not contain references to remote tables).

The forms of subquery predicates include:

- Unquantified comparison predicates:

  ```
  <scalar-expression> <comparison-operator>
  <subquery>
  ```

  The comparison operator is: =, <>, >, >=, <, or <=

  Unquantified comparison subqueries return exactly one value. If the subquery returns more than one value, an error message appears. This type of query is also called a scalar subquery predicate.

- IN predicates:

  ```
  <scalar-expression> [NOT] IN <subquery>
  ```

  The IN subquery predicate returns a list of values or a single value. This type is also called a quantified subquery predicate.

- Existence predicates:

```
[NOT] EXISTS <subquery>
```

The EXISTS predicate represents the existence of the subquery. The expression EXISTS <subquery> evaluates to true only if the subquery result is not empty. The EXISTS predicate does not compare results with any column or expressions in the outer query block, and is typically used with correlated subqueries.

- Quantified comparison predicates:

  ```
  <scalar-expression> <comparison-operator> [ANY |
  ALL] <subquery>
  ```

  A quantified comparison predicate compares one or a collection of values returned from a subquery.

The types of queries you can run include:

- Disjunction of uncorrelated scalar subqueries or IN subqueries that cannot be executed vertically within the WHERE or HAVING clause
- Disjunction of correlated/uncorrelated EXISTS subqueries within the WHERE or HAVING clause
- Disjunction of arbitrary correlated/uncorrelated scalar subqueries, IN or EXISTS subqueries, or quantified comparison subqueries within the WHERE or HAVING clause
- Arbitrary uncorrelated/correlated subquery predicates combined with AND/OR (conjunct/disjunct) and simple predicates or subquery predicates
- Conjunction/disjunction of subquery predicates on top of a view/derived table
- Disjunction of subquery predicates in UPDATE, DELETE, and SELECT INTO statements

The SUBQUERY_CACHING_PREFERENCE option lets experienced DBAs choose which subquery caching method to use. See SUBQUERY_CACHING_PREFERENCE option in *Reference: Statements and Options*.

Examples

Example 1 Disjunction of uncorrelated EXISTS and IN subqueries:

```
SELECT COUNT(*)
FROM supplier
WHERE s_suppkey IN (SELECT MAX(l_suppkey)
                    FROM lineitem
                    GROUP BY l_linenumber)
```

```
OR EXISTS (SELECT p_brand
          FROM part
          WHERE p_brand = 'Brand#43');
```

Example 2 Disjunction of uncorrelated EXISTS subqueries:

```
SELECT COUNT(*)
FROM supplier
WHERE EXISTS (SELECT l_suppkey
             FROM lineitem
             WHERE l_suppkey = 12345)
OR EXISTS    (SELECT p_brand
             FROM part
             WHERE p_brand = 'Brand#43');
```

Example 3 Disjunction of uncorrelated scalar or IN subquery predicates:

```
SELECT COUNT(*)
FROM supplier
WHERE s_acctbal*10 > (SELECT MAX(o_totalprice)
                     FROM orders
                     WHERE o_custkey = 12345)
OR substring(s_name, 1, 6) IN (SELECT c_name
                              FROM Customers
                              WHERE c_nationkey = 10);
```

Example 4 Disjunction of correlated/uncorrelated quantified comparison subqueries:

```
SELECT COUNT(*)
FROM lineitem
WHERE l_suppkey > ANY (SELECT MAX(s_suppkey)
                      FROM supplier
                      WHERE s_acctbal >100
                      GROUP BY s_nationkey)
OR l_partkey >= ANY (SELECT MAX(p_partkey)
                    FROM part
                    GROUP BY p_mfgr);
```

Example 5 Disjunction of any correlated subquery predicates:

```
SELECT COUNT(*)
FROM supplier S
WHERE EXISTS (SELECT l_suppkey
             FROM lineitem
             WHERE l_suppkey = S.s_suppkey)

OR EXISTS (SELECT p_brand FROM part
          WHERE p_brand = 'Brand#43'
            AND p_partkey > S.s_suppkey);
```

Before support for disjunction of subqueries, users were required to write queries in two parts, and then use UNION to merge the final results.

The following query illustrates a merged query that gets the same results as the query in "Example 5" on page 38. Performance of the merged query is suboptimal because it scans the supplier table twice and then merges the results from each UNION to return the final result.

```
SELECT COUNT(*)
FROM (SELECT s_suppkey FROM supplier S
     WHERE EXISTS (SELECT l_suppkey
                   FROM lineitem
                   WHERE l_suppkey = S.s_suppkey)
UNION

SELECT s_suppkey
FROM supplier S
WHERE EXISTS (SELECT p_brand
             FROM part
             WHERE p_brand = 'Brand#43'
               AND p_partkey > S.s_suppkey)) as UD;
```

ALL or ANY conditions

The syntax for ANY conditions is:

expression compare **ANY** (*subquery*)

where *compare* is a comparison operator.

For example, an ANY condition with an equality operator is TRUE if *expression* is equal to any of the values in the result of the subquery, and FALSE if the expression is not NULL and does not equal any of the columns of the subquery:

expression = **ANY** (*subquery*)

The ANY condition is UNKNOWN if *expression* is the NULL value, unless the result of the subquery has no rows, in which case the condition is always FALSE.

You can use the keyword SOME instead of ANY.

The syntax for ALL conditions is:

expression compare **ALL** (*subquery*)

where *compare* is a comparison operator.

Restrictions

If there is more than one expression on either side of a quantified comparison predicate, an error message is returned. For example:

```
Subquery allowed only one select list item
```

Queries of this type can always be expressed in terms of IN subqueries or scalar subqueries using MIN and MAX set functions.

Compatibility

ANY and ALL subqueries are compatible between Adaptive Server Enterprise and Sybase IQ. Only Sybase IQ supports SOME as a synonym for ANY.

BETWEEN conditions

The syntax for BETWEEN conditions is as follows:

expr [**NOT**] **BETWEEN** *start-expr* **AND** *end-expr*

The BETWEEN condition can evaluate as TRUE, FALSE, or UNKNOWN. Without the NOT keyword, the condition evaluates as TRUE if *expr* is between *start-expr* and *end-expr*. The NOT keyword reverses the meaning of the condition but leaves UNKNOWN unchanged.

The BETWEEN condition is equivalent to a combination of two inequalities:

expr >= *start-expr* **AND** *expr* <= *end-expr*

A BETWEEN predicate is of the form "A between B and C." Either "B" or "C" or both "B" and "C" can be subqueries. "A" must be a value expression or column.

Compatibility

The BETWEEN condition is compatible between Sybase IQ and Adaptive Server Enterprise.

LIKE conditions

The syntax for LIKE conditions is:

expression [**NOT**] **LIKE** *pattern* [**ESCAPE** *escape-expr*]

The LIKE condition can evaluate as TRUE, FALSE, or UNKNOWN. You can use LIKE only on string data.

You cannot use subqueries inside a LIKE predicate.

LIKE predicates that start with characters other than wildcard characters may execute faster if an HG or LF index is available.

Certain LIKE predicates execute faster, if a WD index is available.

Without the NOT keyword, the condition evaluates as TRUE if *expression* matches the *pattern*. If either *expression* or *pattern* is the NULL value, this condition is UNKNOWN. The NOT keyword reverses the meaning of the condition but leaves UNKNOWN unchanged.

The pattern might contain any number of wildcard characters. The wildcard characters are:

Wildcard	Matches
_ (underscore)	Any one character
% (percent)	Any string of zero or more characters
[]	Any single character in the specified range or set
[^]	Any single character not in the specified range or set

All other characters must match exactly.

For example, the search condition:

```
name LIKE 'a%b_'
```

is TRUE for any row where name starts with the letter *a* and has the letter *b* as its second-to-last character.

If you specify an *escape-expr*, it must evaluate to a single character. The character can precede a percent, an underscore, a left square bracket, or another escape character in the *pattern* to prevent the special character from having its special meaning. When escaped in this manner, a percent matches a percent, and an underscore matches an underscore.

All patterns of 126 characters or less are supported. Patterns of length greater than 254 characters are not supported. Some patterns of length between 127 and 254 characters are supported, depending on the contents of the pattern.

Searching for one of a set of characters

You can specify a set of characters to look for by listing the characters inside square brackets. For example, the following condition finds the strings *smith* and *smyth*:

```
LIKE 'sm[iy]th'
```

Searching for one of a range of characters

Specify a range of characters to look for by listing the ends of the range inside square brackets, separated by a hyphen. For example, the following condition finds the strings *bough* and *rough*, but not *tough*:

```
LIKE '[a-r]ough'
```

The range of characters [a-z] is interpreted as "greater than or equal to a, and less than or equal to z," where the greater than and less than operations are carried out within the collation of the database. For information on ordering of characters within a collation, see Chapter 11, "International Languages and Character Sets" in the *System Administration Guide: Volume 1*.

The lower end of the range must precede the higher end of the range. For example, a LIKE condition containing the expression [z-a] returns no rows, because no character matches the [z-a] range.

Unless the database is created as case-sensitive, the range of characters is case-insensitive. For example, the following condition finds the strings *Bough*, *rough*, and *TOUGH*:

```
LIKE '[a-z]ough'
```

If the database is created as a case-sensitive database, the search condition is case-sensitive also.

Combining searches for ranges and sets

You can combine ranges and sets within square brackets. For example, the following condition finds the strings *bough*, *rough*, and *tough*:

```
LIKE '[a-rt]ough'
```

The bracket *[a-mpqs-z]* is interpreted as "exactly one character that is either in the range *a* to *m* inclusive, or is *p*, or is *q*, or is in the range *s* to *z* inclusive."

Searching for one character not in a range

Use the caret character (^) to specify a range of characters that is excluded from a search. For example, the following condition finds the string *tough*, but not the strings *rough*, or *bough*:

```
LIKE '[^a-r]ough'
```

The caret negates the entire contents of the brackets. For example, the bracket *[^a-mpqs-z]* is interpreted as "exactly one character that is not in the range *a* to *m* inclusive, is not *p*, is not *q*, and is not in the range *s* to *z* inclusive."

Special cases of ranges and sets

Any single character in square brackets indicates that character. For example, *[a]* matches just the character *a*. *[^]* matches just the caret character, *[%]* matches only the percent character (the percent character does not act as a wildcard character in this context), and *[_]* matches just the underscore character. Also, *[[]* matches only the character *[*.

Other special cases are:

- The expression *[a-]* matches either of the characters *a* or -.
- The expression *[]* is never matched and always returns no rows.

- The expressions *[* or *[abp-q* are ill-formed expressions, and give syntax errors.
- You cannot use wildcard characters inside square brackets. The expression *[a%b]* finds one of *a*, *%*, or *b*.
- You cannot use the caret character to negate ranges except as the first character in the bracket. The expression *[a^b]* finds one of *a*, *^*, or *b*.

Compatibility

The ESCAPE clause is supported by Sybase IQ only.

IN conditions

The syntax for IN conditions is:

{ *expression* [**NOT**] **IN** (*subquery*)
| *expression* [**NOT**] **IN** (*expression*)
| *expression* [**NOT**] **IN** (*value-expr1* , *value-expr2*
[, *value-expr3*] …) }

Without the NOT keyword, the IN condition is TRUE if *expression* equals any of the listed values, UNKNOWN if *expression* is the NULL value, and FALSE otherwise. The NOT keyword reverses the meaning of the condition but leaves UNKNOWN unchanged.

The maximum number of values allowed in an IN condition list is 250,000.

Compatibility

IN conditions are compatible between Adaptive Server Enterprise and Sybase IQ.

CONTAINS conditions

The syntax for CONTAINS conditions is as follows:

{ *column-name* [**NOT**] **CONTAINS** ((*word1* [, *word2*] [, *word3*] …)

The *column-name* must be a CHAR, VARCHAR, or LONG VARCHAR (CLOB) column in a base table, and must have a WD index. The *word1*, *word2* and *word3* expressions must be string constants no longer than 255 bytes, each containing exactly one word. The length of that word cannot exceed the maximum permitted word length of the word index of the column.

Without the NOT keyword, the CONTAINS condition is TRUE if *column-name* contains each of the words, UNKNOWN if *column-name* is the NULL value, and FALSE otherwise. The NOT keyword reverses these values but leaves UNKNOWN unchanged.

For example, this search condition:

```
varchar_col CONTAINS ('cat', ‘mat’)
```

is TRUE if the value of *varchar_col* is `The cat is on the mat`. If the value of *varchar_col* is `The cat chased the mouse`, this condition is FALSE.

When Sybase IQ executes a statement containing both LIKE and CONTAINS, the CONTAINS condition takes precedence.

Avoid using the CONTAINS predicate in a view that has a user-defined function, because the CONTAINS criteria are ignored. Use the LIKE predicate with wildcards instead, or issue the query outside of a view.

EXISTS conditions

The syntax for EXISTS conditions is as follows:

EXISTS(*subquery*)

The EXISTS condition is TRUE if the subquery result contains at least one row, and FALSE if the subquery result does not contain any rows. The EXISTS condition cannot be UNKNOWN.

Compatibility

The EXISTS condition is compatible between Adaptive Server Enterprise and Sybase IQ.

IS NULL conditions

The syntax for IS NULL conditions is:

expression **IS** [**NOT**] **NULL**

Without the NOT keyword, the IS NULL condition is TRUE if the expression is the NULL value, and FALSE otherwise. The NOT keyword reverses the meaning of the condition.

Compatibility

The IS NULL condition is compatible between Adaptive Server Enterprise and Sybase IQ.

Conditions with logical operators

Search conditions can be combined using AND, OR, and NOT.

Conditions are combined using AND as follows:

condition1 **AND** *condition2*

If both conditions are TRUE, the combined condition is TRUE. If either condition is FALSE, the combined condition is FALSE. If otherwise, the combined condition is UNKNOWN.

Conditions are combined using OR as follows:

condition1 **OR** *condition2*

If both conditions are TRUE, the combined condition is TRUE. If either condition is FALSE, the combined condition is FALSE. If otherwise, the combined condition is UNKNOWN. There is no guaranteed order as to which condition, *condition1* or *condition2*, is evaluated first.

See “Disjunction of subquery predicates” on page 36.

Compatibility

The AND and OR operators are compatible between Sybase IQ and Adaptive Server Enterprise.

NOT conditions

The syntax for NOT conditions is:

NOT *condition1*

The NOT condition is TRUE if *condition1* is FALSE, FALSE if *condition1* is TRUE, and UNKNOWN if *condition1* is UNKNOWN.

Truth value conditions

The syntax for truth value conditions is:

IS [**NOT**] *truth-value*

Without the NOT keyword, the condition is TRUE if the *condition* evaluates to the supplied *truth-value*, which must be one of TRUE, FALSE, or UNKNOWN. Otherwise, the value is FALSE. The NOT keyword reverses the meaning of the condition but leaves UNKNOWN unchanged.

Compatibility

Truth-valued conditions are supported by Sybase IQ only.

Three-valued logic

The following tables show how the AND, OR, NOT, and IS logical operators of SQL work in three-valued logic.

AND operator

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR operator

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT operator

TRUE	FALSE	UNKNOWN
FALSE	TRUE	UNKNOWN

IS operator

IS	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE
UNKNOWN	FALSE	FALSE	TRUE

User-supplied condition hints

The Sybase IQ query optimizer uses information from available indexes to select an appropriate strategy for executing a query. For each condition in the query, the optimizer decides whether the condition can be executed using indexes, and if so, the optimizer chooses which index and in what order with respect to the other conditions on that table. The most important factor in these decisions is the selectivity of the condition; that is, the fraction of the table's rows that satisfy that condition.

The optimizer normally decides without user intervention, and it generally makes optimal decisions. In some situations, however, the optimizer might not be able to accurately determine the selectivity of a condition before it has been executed. These situations normally occur only where either the condition is on a column with no appropriate index available, or where the condition involves some arithmetic or function expression and is, therefore, too complex for the optimizer to accurately estimate.

If you have a query that is run frequently, then you may want to experiment to see whether you can improve the performance of that query by supplying the optimizer with additional information to aid it in selecting the optimal execution strategy.

User-supplied condition selectivity

The simplest form of condition hint is to supply a selectivity value that will be used instead of the value the optimizer would have computed.

Selectivity hints are supplied within the text of the query by wrapping the condition within parentheses. Then within the parentheses, after the condition, you add a comma and a numeric value to be used as the selectivity.

This selectivity value is expressed as a percentage of the table's rows, which satisfy the condition. Possible numeric values for selectivity thus range from 100.0 to 0.0.

Note In query plans, selectivity is expressed as a fraction instead of as a percentage; so a user-supplied selectivity of 35.5 appears in that query's plan as a selectivity of 0.355000.

Examples

- The following query provides an estimate that one and one half percent of the ship_date values are earlier than 1994/06/30:

```
SELECT  ShipDate
FROM  SalesOrderItems
WHERE ( ShipDate < '2001/06/30', 1.5 )
ORDER BY ShipDate DESC
```

- The following query estimates that half a percent of the rows satisfy the condition:

```
SELECT *
FROM Customers c, SalesOrders o
WHERE (o.SalesRepresentative > 1000.0, 0.5)
   AND c.ID = o.customerID
```

Fractional percentages enable more precise user estimates to be specified and can be particularly important for large tables.

Compatibility

SQL Anywhere Studio supports user-supplied selectivity estimates.

Adaptive Server Enterprise does not support user-supplied selectivity estimates.

User-supplied condition hint strings

In addition to supporting user-supplied selectivity estimates, Sybase IQ also lets users supply additional hint information to the optimizer through a condition hint string. These per-condition hint strings let users specify additional execution preferences for a condition, which the optimizer follows, if possible. These preferences include which index to use for the condition, the selectivity of the condition, the phase of execution when the condition is executed, and the usefulness of the condition, which affects its ordering among the set of conditions executed within one phase of execution.

Condition hint strings, like the user-supplied selectivity estimates, are supplied within the text of the query by wrapping the condition within parentheses. Then within the parentheses and after the condition, you add a comma and a supply a quoted string containing the desired hints. Within that quoted string each hint appears as a hint type identifier, followed by a colon and the value for that hint type. Multiple hints within the same hint string are separated from each other by a comma, and multiple hints can appear in any order. White space is allowed between any of two elements within a hint string.

There are four different supported hint types:

- Selectivity hints, which are equivalent to the user-supplied selectivity estimates
- Index preference hints
- Execution phase hints
- Usefulness hints

Selectivity hints

The first hint type that can appear within a hint string is a selectivity hint. A selectivity hint is identified by a hint type identifier of either “S” or “s”. Like user-supplied selectivity estimates, the selectivity value is always expressed as a percentage of the table’s rows, which satisfy the condition.

Example

The following example is exactly equivalent to the second example in “User-supplied condition selectivity” on page 47.

```
SELECT *
FROM Customers c, SalesOrders o
WHERE (o.SalesRepresentative > 1000.0, 's: 0.5)
   AND c.ID = o.CustomerID
```

Index preference hints

The next supported hint type is an index preference hint, which is identified by a hint type identifier of either “I” or “i”. The value for an index preference hint can be any integer between -10 and 10. The meaning of each positive integer value is to prefer a specific index type, while negative values indicate that the specific index type is to be avoided.

The effect of an index preference hint is the same as that of the INDEX_PREFERENCE option, except that the preference applies only to the condition it is associated with rather than all conditions within the query. An index preference can only affect the execution of a condition if the specified index type exists on that column and that index type is valid for use when evaluating the associated condition; not all index types are valid for use with all conditions. See “INDEX_PREFERENCE option,” in Chapter 2, “Database Options,” in *Reference: Statements and Options* for the specific meanings of integers between -10 and 10.

Example

The following example specifies a 3 percent selectivity and indicates that, if possible, the condition should be evaluated using an HG index:

```
SELECT *
FROM Customers c, SalesOrders o
WHERE (o.SalesRepresentative > 1000.0, 'S:3.00, I:+2')
   AND c.ID = o.CustomerID
```

The next example specifies a 37.5 percent selectivity and indicates that if possible the condition should not be evaluated using an HG index:

```
SELECT *
FROM Customers c, SalesOrders o
WHERE (o.SalesRepresentative > 1000.0, 'i:-2,
s:37.500')
   AND c.ID = o.CustomerID
```

Execution phase hints

The third supported hint type is the execution phase hint, which is identified with a hint type identifier of either “E” or “e”.

Within the Sybase IQ query engine, there are distinct phases of execution where conditions can be evaluated: invariant, delayed, bound, and horizontal.

By default, the optimizer chooses to evaluate each condition within the earliest phase of execution where all the information needed to evaluate that condition is available. Every condition. therefore, has a default execution phase where it is evaluated.

Because no condition can be evaluated before the information it needs is available, the execution phase hint can only be used to delay the execution of a condition to a phase after its default phase. It cannot be used to force a condition to be evaluated within any phase earlier than its default phase.

The four phases of condition execution from earliest to latest are described as follows:

Invariant A condition that refers to only one column (or two columns from the same table) and that can be evaluated using an index is generally referred to as a simple invariant condition. Simple invariant condition are normally evaluated early within the optimization process.

This means that the number of rows satisfying all of those invariant conditions is available to guide the optimizer's decisions on the best join order and join algorithms to use. Because this is the earliest phase of execution, a user can never force a condition into this phase, but conditions can be forced out of this phase into later phases.

Delayed Some conditions cannot be evaluated until some other part of a query has been executed. These delayed conditions are evaluated once when the query node to which they are attached is first fetched. These conditions fall into two categories, uncorrelated subquery conditions and IN or PROBABLY_IN pushdown join conditions created by the optimizer.

Bound Some conditions must be evaluated multiple times. These conditions generally fall into two categories: conditions containing outer references within a correlated subquery, and pushdown equality join conditions created by the optimizer. The outer reference conditions, for example, are reevaluated each time the outer reference value changes during the query's execution.

Horizontal Some conditions, such as those which contain more than two columns from a table, must be evaluated one row at a time, rather than by using an index.

An execution phase hint accepts a values that identifies in which execution phase the user wants the condition to be evaluated. Each value is a case-insensitive single character:

- D – Delayed

- B – Bound
- H – Horizontal

Example

The following example shows a condition hint string which indicates that the condition should be moved into the “Delayed” phase of execution, and it indicates that if possible the condition should be evaluated using an LF index.:

```
SELECT *
FROM Customers c, SalesOrders o
WHERE (o.SalesRepresentative > 10000.0, 'E:D, I:1')
   AND c.id = o.CustomerID
```

Usefulness hints

The final supported hint type is the usefulness hint, which is identified by a hint type identifier of either “U” or “u”. The value for a usefulness hint can be any numeric value between 0.0 and 10.0. Within the optimizer a usefulness value is computed for every condition, and the usefulness value is then used to determine the order of evaluation among the set of conditions to be evaluated within the same phase of execution. The higher the usefulness value, the earlier it appears in the order of evaluation. Supplying a usefulness hint lets users place a condition at a particular point within the order of evaluation, but it cannot change the execution phase within which the condition is evaluated.

Example

The following example shows a condition hint string which indicates that the condition should be moved into the “Delayed” phase of execution, and that its usefulness should be set to 3.25 within that “Delayed” phase.

```
SELECT *
FROM Customers c, SalesOrders o
WHERE (co.SalesRepresentative > 10000.0, 'U: 3.25,  E:
D')
AND c.id = o.CustomerID
```

Compatibility

SQL Anywhere Studio does not support user-supplied condition hint strings.

Adaptive Server Enterprise does not support user-supplied condition hint strings.

User-supplied hints on join equality conditions

Users can specify a join algorithm preference that does not affect every join in the query.

Simple equality join predicates can be tagged with a predicate hint that allows a join preference to be specified for just that one join. If the same join has more than one join condition with a local join preference, and if those hints are not the same value, then all local preferences are ignored for that join. Local join preferences do not affect the join order chosen by the optimizer.

The following example requests a hash join:

```
AND (T.X = 10 * R.x, 'J:4')
```

Guidelines for usage of user-supplied condition hints

Condition hints are generally appropriate only within frequently run queries.

Only advanced users should experiment with condition hints. The optimizer generally makes optimal decisions, except where it cannot infer accurate information about a condition from the available indexes.

The optimizer often rewrites or simplifies the original conditions, and it also infers new conditions from the original conditions. Condition hints are not carried through new to conditions inferred by the optimizer, nor are they carried through to simplified conditions.

Special values

Special values can be used in expressions, and as column defaults when creating tables.

CURRENT DATABASE special value

Function
CURRENT DATABASE returns the name of the current database.

Data type
STRING

See also
"Expressions" on page 23

CURRENT DATE special value

Function
The current year, month and day.

Data type
DATE

See also
"Expressions" on page 23

"Date and time data types" on page 82

CURRENT PUBLISHER special value

Function
CURRENT PUBLISHER returns a string that contains the publisher user ID of the database for SQL Remote replications.

Data type
STRING

CURRENT PUBLISHER can be used as a default value in columns with character data types.

See also
"Expressions" on page 23

CURRENT TIME special value

Function
The current hour, minute, second, and fraction of a second.

Data type
TIME

Description — The fraction of a second is stored to 6 decimal places, but the accuracy of the current time is limited by the accuracy of the system clock.

See also — "Expressions" on page 23

"Date and time data types" on page 82

CURRENT TIMESTAMP special value

Function — Combines CURRENT DATE and CURRENT TIME to form a TIMESTAMP value containing the year, month, day, hour, minute, second and fraction of a second. As with CURRENT TIME, the accuracy of the fraction of a second is limited by the system clock.

CURRENT TIMESTAMP defaults to 3 digits.

Data type — TIMESTAMP

See also — "Expressions" on page 23

"Date and time data types" on page 82

CURRENT USER special value

Function — CURRENT USER returns a string that contains the user ID of the current connection.

Data type — STRING

CURRENT USER can be used as a default value in columns with character data types.

Description — On UPDATE, columns with a default value of CURRENT USER are not changed.

See also — "Expressions" on page 23

LAST USER special value

Function — The name of the user who last modified the row.

Data type — STRING

	LAST USER can be used as a default value in columns with character data types.
Description	On INSERT and LOAD, this constant has the same effect as CURRENT USER. On UPDATE, if a column with a default value of LAST USER is not explicitly modified, it is changed to the name of the current user.
	When combined with the DEFAULT TIMESTAMP, a default value of LAST USER can be used to record (in separate columns) both the user and the date and time a row was last changed.
See also	"CURRENT USER special value" on page 54
	"CURRENT TIMESTAMP special value" on page 54
	CREATE TABLE statement in *Reference: Statements and Options*

SQLCODE special value

Function	Current SQLCODE value.
DATA TYPE	STRING
DESCRIPTION	The SQLCODE value is set after each statement. You can check the SQLCODE to see whether or not the statement succeeded.
See also	"Expressions" on page 23
	Chapter 1, "SQL Statements," in *System Administration Guide: Volume 2*

SQLSTATE special value

Function	Current SQLSTATE value.
Data type	STRING
Description	The SQLSTATE value is set after each statement. You can check the SQLSTATE to see whether or not the statement succeeded.
See also	"Expressions" on page 23
	Chapter 1, "SQL Statements," in *System Administration Guide: Volume 2*

TIMESTAMP special value

Function

TIMESTAMP indicates when each row in the table was last modified.

Data type

TIMESTAMP

Description

When a column is declared with DEFAULT TIMESTAMP, a default value is provided for insert and load operations. The value is updated with the current date and time whenever the row is updated.

On INSERT and LOAD, DEFAULT TIMESTAMP has the same effect as CURRENT TIMESTAMP. On UPDATE, if a column with a default value of TIMESTAMP is not explicitly modified, the value of the column is changed to the current date and time.

Note Sybase IQ does not support DEFAULT values of UTC TIMESTAMP or CURRENT UTC TIMESTAMP, nor does IQ support the database option DEFAULT_TIMESTAMP_INCREMENT. Sybase IQ generates an error every time an attempt is made to insert or update the DEFAULT value of a column of type UTC TIMESTAMP or CURRENT UTC TIMESTAMP.

See also

"Date and time data types" on page 82

USER special value

Function

USER returns a string that contains the user ID of the current connection.

Data type

STRING

USER can be used as a default value in columns with character data types.

Description

On UPDATE, columns with a default value of USER are not changed.

See also

"Expressions" on page 23

"CURRENT USER special value" on page 54

"LAST USER special value" on page 54

Variables

Sybase IQ supports three levels of variables:

- **Local variables** These are defined inside a compound statement in a procedure or batch using the DECLARE statement. They exist only inside the compound statement.
- **Connection-level variables** These are defined with a CREATE VARIABLE statement. They belong to the current connection, and disappear when you disconnect from the database or when you use the DROP VARIABLE statement.
- **Global variables** These are variables that have system-supplied values.

Local and connection-level variables are declared by the user, and can be used in procedures or in batches of SQL statements to hold information. Global variables are system-supplied variables that provide system-supplied values. All global variables have names beginning with two @ signs. For example, the global variable *@@version* has a value that is the current version number of the database server. Users cannot define global variables.

Local variables

Local variables are declared using the DECLARE statement, which can be used only within a compound statement (that is, bracketed by the BEGIN and END keywords). The variable is initially set as NULL. You can set the value of the variable using the SET statement, or you can assign the value using a SELECT statement with an INTO clause.

The syntax of the DECLARE statement is as follows:

```
DECLARE variable-name data-type
```

You can pass local variables as arguments to procedures, as long as the procedure is called from within the compound statement.

Examples

- The following batch illustrates the use of local variables:

  ```
  BEGIN
  DECLARE local_var INT ;
  SET local_var = 10 ;
  MESSAGE 'local_var = ', local_var ;
  END
  ```

  Running this batch from ISQL displays this message on the server window:

  ```
  local_var = 10
  ```

- The variable local_var does not exist outside the compound statement in which it is declared. The following batch is invalid, and displays a column not found error:

  ```
  -- This batch is invalid.
  BEGIN
  DECLARE local_var INT ;
  SET local_var = 10 ;
  MESSAGE 'local_var = ', local_var ;
  END;
  MESSAGE 'local_var = ', local_var ;
  ```

- The following example illustrates the use of SELECT with an INTO clause to set the value of a local variable:

  ```
  BEGIN
  DECLARE local_var INT ;
  SELECT 10 INTO local_var ;
  MESSAGE 'local_var = ', local_var ;
  END
  ```

  Running this batch from ISQL displays this message on the server window:

  ```
  local_var = 10
  ```

Compatibility

- **Names** Adaptive Server Enterprise and Sybase IQ both support local variables. In Adaptive Server Enterprise, all variables must be prefixed with an @ sign. In Sybase IQ, the @ prefix is optional. To write compatible SQL, ensure all your variables have the @ prefix.
- **Scope** The scope of local variables differs between Sybase IQ and Adaptive Server Enterprise. Sybase IQ supports the use of the DECLARE statement to declare local variables within a batch. However, if the DECLARE is executed within a compound statement, the scope is limited to the compound statement.
- **Declaration** Only one variable can be declared for each DECLARE statement in Sybase IQ. In Adaptive Server Enterprise, more than one variable can be declared in a single statement.

Connection-level variables

Connection-level variables are declared with the CREATE VARIABLE statement. The CREATE VARIABLE statement can be used anywhere except inside compound statements. Connection-level variables can be passed as parameters to procedures.

The syntax for CREATE VARIABLE is:

```
CREATE VARIABLE variable-name data-type
```

When a variable is created, it is initially set to NULL. You can set the value of connection-level variables in the same way as local variables, using the SET statement or using a SELECT statement with an INTO clause.

Connection-level variables exist until the connection is terminated, or until you explicitly drop the variable using the DROP VARIABLE statement. The following statement drops the variable *con_var*:

```
DROP VARIABLE con_var
```

Example

- The following batch of SQL statements illustrates the use of connection-level variables.

  ```
  CREATE VARIABLE con_var INT;
  SET con_var = 10;
  MESSAGE 'con_var = ', con_var;
  ```

  Running this batch from ISQL displays this message on the server window:

  ```
  con_var = 10
  ```

Compatibility

Adaptive Server Enterprise does not support connection-level variables.

Global variables

Global variables have values set by Sybase IQ. For example, the global variable *@@version* has a value that is the current version number of the database server.

Global variables are distinguished from local and connection-level variables by two @ signs preceding their names. For example, *@@error* is a global variable. Users cannot create global variables, and cannot update the value of global variables directly.

Some global variables, such as *@@spid*, hold connection-specific information and therefore have connection-specific values. Other variables, such as *@@connections*, have values that are common to all connections.

Global variable and special constants

The special constants such as CURRENT DATE, CURRENT TIME, USER, SQLSTATE, and so on are similar to global variables.

The following statement retrieves the value of the version global variable:

```
SELECT @@version
```

In procedures, global variables can be selected into a variable list. The following procedure returns the server version number in the *ver* parameter.

```
CREATE PROCEDURE VersionProc ( OUT ver
                 VARCHAR ( 100) )
BEGIN
    SELECT @@version
    INTO ver;
END
```

In Embedded SQL, global variables can be selected into a host variable list.

List of global variables

Table 2-5 lists the global variables available in Sybase IQ.

Table 2-5: Sybase IQ global variables

Variable name	Meaning
@@error	Commonly used to check the error status (succeeded or failed) of the most recently executed statement. Contains 0 if the previous transaction succeeded; otherwise, contains the last error number generated by the system. A statement such as if @@error != 0 return causes an exit if an error occurs. Every SQL statement resets @@error, so the status check must immediately follow the statement whose success is in question.
@@fetch_status	Contains status information resulting from the last fetch statement. @@fetch_status may contain the following values • 0 The fetch statement completed successfully. • -1 The fetch statement resulted in an error. • -2 There is no more data in the result set. This feature is the same as @@sqlstatus, except that it returns different values. It is for Microsoft SQL Server compatibility.
@@identity	The last value inserted into an Identity/Autoincrement column by an insert, load or update statement. *@@identity* is reset each time a row is inserted into a table. If a statement inserts multiple rows, *@@identity* reflects the Identity/Autoincrement value for the last row inserted. If the affected table does not contain an Identity/Autoincrement column, *@@identity* is set to 0. The value of *@@identity* is not affected by the failure of an insert, load, or update statement, or the rollback of the transaction that contained the failed statement. *@@identity* retains the last value inserted into an Identity/Autoincrement column, even if the statement that inserted that value fails to commit.
@@isolation	Current isolation level. @@isolation takes the value of the active level.
@@procid	Stored procedure ID of the currently executing procedure.
@@servername	Name of the current database server.
@@sqlstatus	Contains status information resulting from the last FETCH statement.
@@version	Version number of the current version of Sybase IQ.

Compatibility

Table 2-6 includes all Adaptive Server Enterprise global variables that are supported in Sybase IQ. Adaptive Server Enterprise global variables that are not supported by Sybase IQ are not included in the list. In contrast to Table 2-5, this list includes all global variables that return a value, including those for which the value is fixed at NULL, 1, -1, or 0, and might not be meaningful.

Table 2-6: ASE global variables supported in Sybase IQ

Global variable	Returns
@@char_convert	Returns 0.
@@client_csname	In Adaptive Server Enterprise, the client's character set name. Set to NULL if client character set has never been initialized; otherwise, contains the name of the most recently used character set. Returns NULL in Sybase IQ.
@@client_csid	In Adaptive Server Enterprise, the client's character set ID. Set to -1 if client character set has never been initialized; otherwise, contains the most recently used client character set ID from syscharsets. Returns -1 in Sybase IQ.
@@connections	The number of logins since the server was last started.
@@cpu_busy	In Adaptive Server Enterprise, the amount of time, in ticks, that the CPU has spent performing Adaptive Server Enterprise work since the last time Adaptive Server Enterprise was started. In Sybase IQ, returns 0.
@@error	Commonly used to check the error status (succeeded or failed) of the most recently executed statement. Contains 0 if the previous transaction succeeded; otherwise, contains the last error number generated by the system. A statement such as: `if @@error != 0 return` causes an exit if an error occurs. Every statement resets *@@error*, including PRINT statements or IF tests, so the status check must immediately follow the statement whose success is in question.
@@identity	In Adaptive Server Enterprise, the last value inserted into an IDENTITY column by an INSERT, LOAD, or SELECT INTO statement. *@@identity* is reset each time a row is inserted into a table. If a statement inserts multiple rows, *@@identity* reflects the IDENTITY value for the last row inserted. If the affected table does not contain an IDENTITY column, *@@identity* is set to 0. The value of *@@identity* is not affected by the failure of an INSERT or SELECT INTO statement, or the rollback of the transaction that contained the failed statement. *@@identity* retains the last value inserted into an IDENTITY column, even if the statement that inserted that value fails to commit.
@@idle	In Adaptive Server Enterprise, the amount of time, in ticks, that Adaptive Server Enterprise has been idle since the server was last started. In Sybase IQ, returns 0.

Global variable	Returns
@@io_busy	In Adaptive Server Enterprise, the amount of time, in ticks, that Adaptive Server Enterprise has spent performing input and output operations since the server was last started. In Sybase IQ, returns 0.
@@isolation	Current isolation level of the connection. In Adaptive Server Enterprise, *@@isolation* takes the value of the active level.
@@langid	In Adaptive Server Enterprise, defines the local language ID of the language currently in use. In Sybase IQ, returns 0.
@@language	In Adaptive Server Enterprise, defines the name of the language currently in use. In Sybase IQ, returns “English”.
@@maxcharlen	In Adaptive Server Enterprise, maximum length, in bytes, of a character in the Adaptive Server Enterprise default character set. In Sybase IQ, returns 1.
@@max_connections	For the network server, the maximum number of active clients (not database connections, as each client can support multiple connections). For Adaptive Server Enterprise, the maximum number of connections to the server.
@@ncharsize	In Adaptive Server Enterprise, average length, in bytes, of a national character. In Sybase IQ, returns 1.
@@nestlevel	In Adaptive Server Enterprise, nesting level of current execution (initially 0). Each time a stored procedure or trigger calls another stored procedure or trigger, the nesting level is incremented. In Sybase IQ, returns -1.
@@pack_received	In Adaptive Server Enterprise, number of input packets read by Adaptive Server Enterprise since the server was last started. In Sybase IQ, returns 0.
@@pack_sent	In Adaptive Server Enterprise, number of output packets written by Adaptive Server Enterprise since the server was last started. In Sybase IQ, returns 0.
@@packet_errors	In Adaptive Server Enterprise, number of errors that have occurred while Adaptive Server Enterprise was sending and receiving packets. In Sybase IQ, returns 0.
@@procid	Stored procedure ID of the currently executing procedure.
@@servername	Name of the local Adaptive Server Enterprise or Sybase IQ server.
@@spid	In Adaptive Server Enterprise, server process ID number of the current process. In Sybase IQ, the connection handle for the current connection. This is the same value as that displayed by the sa_conn_info procedure.

Global variable	Returns
@@sqlstatus	Contains status information resulting from the last FETCH statement. *@@sqlstatus* may contain the following values: • 0 – the FETCH statement completed successfully. • 1 – the FETCH statement resulted in an error. • 2 – there is no more data in the result set.
@@thresh_hysteresis	In Adaptive Server Enterprise, change in free space required to activate a threshold. In Sybase IQ, returns 0.
@@timeticks	In Adaptive Server Enterprise, number of microseconds per tick. The amount of time per tick is machine-dependent. In Sybase IQ, returns 0.
@@total_errors	In Adaptive Server Enterprise, number of errors that have occurred while Adaptive Server Enterprise was reading or writing. In Sybase IQ, returns 0.
@@total_read	In Adaptive Server Enterprise, number of disk reads by Adaptive Server Enterprise since the server was last started. In Sybase IQ, returns 0.
@@total_write	In Adaptive Server Enterprise, number of disk writes by Adaptive Server Enterprise since the server was last started. In Sybase IQ, returns 0.
@@tranchained	Current transaction mode of the Transact-SQL program. *@@tranchained* returns 0 for unchained or 1 for chained.
@@trancount	Nesting level of transactions. Each BEGIN TRANSACTION in a batch increments the transaction count.
@@transtate	In Adaptive Server Enterprise, current state of a transaction after a statement executes. In Sybase IQ, returns -1.
@@version	Information about the current version of Adaptive Server Enterprise or Sybase IQ.

Comments

Use comments to attach explanatory text to SQL statements or statement blocks. The database server does not execute comments.

Several comment indicators are available in Sybase IQ:

- **-- (Double hyphen)** The database server ignores any remaining characters on the line. This is the SQL92 comment indicator.

- **// (Double slash)** The double slash has the same meaning as the double hyphen.
- **/* … */ (Slash-asterisk)** Any characters between the two comment markers are ignored. The two comment markers might be on the same or different lines. Comments indicated in this style can be nested. This style of commenting is also called C-style comments.
- **% (Percent sign)** The percent sign has the same meaning as the double hyphen. Sybase recommends that you do not use % as a comment indicator.

Note The double-hyphen and the slash-asterisk comment styles are compatible with Adaptive Server Enterprise.

Examples

- This example illustrates the use of double-dash comments:

```
CREATE FUNCTION fullname (firstname CHAR(30),
          lastname CHAR(30))
RETURNS CHAR(61)
-- fullname concatenates the firstname and lastname
-- arguments with a single space between.
BEGIN
   DECLARE name CHAR(61);
   SET name = firstname || ' ' || lastname;
   RETURN ( name );
END
```

- This example illustrates the use of C-style comments:

```
/*
   Lists the names and employee IDs of employees
   who work in the sales department.
*/
CREATE VIEW SalesEmployee AS
SELECT emp_id, emp_lname, emp_fname
FROM "GROUPO".Employees
WHERE DepartmentID = 200
```

NULL value

Function

To specify a value that is unknown or not applicable

Syntax

NULL

Usage

Anywhere

Permissions

Must be connected to the database

Side effects

None

Description

The NULL value is a special value that is different from any valid value for any data type. However, the NULL value is a legal value in any data type. The NULL value is used to represent missing or inapplicable information. These are two separate and distinct cases where NULL is used:

Situation	Description
missing	The field does have a value, but that value is unknown.
inapplicable	The field does not apply for this particular row.

SQL allows columns to be created with the NOT NULL restriction. This means that those particular columns cannot contain the NULL value.

The NULL value introduces the concept of three valued logic to SQL. The NULL value compared using any comparison operator with any value including the NULL value is UNKNOWN. The only search condition that returns TRUE is the IS NULL predicate. In SQL, rows are selected only if the search condition in the WHERE clause evaluates to TRUE; rows that evaluate to UNKNOWN or FALSE are not selected.

You can also use the IS [NOT] *truth-value* clause, where *truth-value* is one of TRUE, FALSE or UNKNOWN, to select rows where the NULL value is involved. See “Search conditions” on page 33 for a description of this clause.

In the following examples, the column Salary contains the NULL value.

Condition	Truth value	Selected?
Salary = NULL	UNKNOWN	NO
Salary <> NULL	UNKNOWN	NO
NOT (Salary = NULL)	UNKNOWN	NO
NOT (Salary <> NULL)	UNKNOWN	NO
Salary = 1000	UNKNOWN	NO
Salary IS NULL	TRUE	YES
Salary IS NOT NULL	FALSE	NO

Condition	Truth value	Selected?
Salary = 1000 IS UNKNOWN	TRUE	YES

The same rules apply when comparing columns from two different tables. Therefore, joining two tables together does not select rows where any of the columns compared contain the NULL value.

The NULL value also has an interesting property when used in numeric expressions. The result of *any* numeric expression involving the NULL value is the NULL value. This means that if the NULL value is added to a number, the result is the NULL value—not a number. If you want the NULL value to be treated as 0, you must use the ISNULL(*expression*, 0) function. See Chapter 4, “SQL Functions.”

Many common errors in formulating SQL queries are caused by the behavior of NULL. Be careful to avoid these problem areas. See “Search conditions” on page 33 for a description of the effect of three-valued logic when combining search conditions.

Example

The following INSERT statement inserts a NULL into the date_returned column of the Borrowed_book table.

```
INSERT
INTO Borrowed_book
( date_borrowed, date_returned, book )
VALUES ( CURRENT DATE, NULL, '1234' )
```

CHAPTER 3 SQL Data Types

About this chapter

This chapter describes the data types supported by Sybase IQ.

Contents

Character data types

Description

For storing strings of letters, numbers and symbols.

Syntax

CHAR [(*max-length*)]

CHARACTER [(*max-length*)]

CHARACTER VARYING [(*max-length*)]

VARCHAR [(*max-length*)]

UNIQUEIDENTIFIERSTR

Usage

CHAR Character data of maximum length *max-length* bytes. If *max-length* is omitted, the default is 1. The maximum size allowed is 32KB – 1. See Notes for restrictions on CHAR data greater than 255 bytes.

See the notes below on character data representation in the database, and on storage of long strings.

All CHAR values are blank padded up to *max-length*, regardless of whether the BLANK PADDING option is specified. When multibyte character strings are held as a CHAR type, the maximum length is still in bytes, not characters.

CHARACTER Same as CHAR.

CHARACTER VARYING Same as VARCHAR.

LONG VARCHAR Arbitrary length character data. The maximum size is limited by the maximum size of the database file (currently 2 gigabytes).

TEXT This is a user-defined data type. It is implemented as a LONG VARCHAR allowing NULL.

VARCHAR Same as CHAR, except that no blank padding is added to the storage of these strings, and VARCHAR strings can have a maximum length of (32KB – 1). See Notes for restrictions on VARCHAR data greater than 255 bytes.

UNIQUEIDENTIFIERSTR Domain implemented as CHAR(36). This data type is used for remote data access, when mapping Microsoft SQL Server uniqueidentifier columns.

Notes

As a separately licensed option, Sybase IQ supports Character Large Object (CLOB) data with a length ranging from zero (0) to 512TB (terabytes) for an IQ page size of 128KB or 2PB (petabytes) for an IQ page size of 512KB. The maximum length is equal to 4GB multiplied by the database page size. See *Large Objects Management in Sybase IQ*.

Storage sizes

Table 3-1 lists the storage size of character data.

Table 3-1: Storage size of character data

Data type	Column definition	Input data	Storage
CHARACTER, CHAR	width of (32K – 1) bytes	(32K – 1) bytes	(32K – 1) bytes
VARCHAR, CHARACTER VARYING	width of (32K – 1) bytes	(32K – 1) bytes	(32K – 1) bytes

Character sets and code pages

Character data is placed in the database using the exact binary representation that is passed from the application. This usually means that character data is stored in the database with the binary representation of the character set used by your system. You can find documentation about character sets in the documentation for your operating system.

On Windows, code pages are the same for the first 128 characters. If you use special characters from the top half of the code page (accented international language characters), you must be careful with your databases. In particular, if you copy the database to a different machine using a different code page, those special characters are retrieved from the database using the original code page representation. With the new code page, they appear on the screen to be the wrong characters.

This problem also appears if you have two clients using the same multiuser server, but running with different code pages. Data inserted or updated by one client might appear incorrect to another.

This problem also shows up if a database is used across platforms. PowerBuilder and many other Windows applications insert data into the database in the standard ANSI character set. If non-Windows applications attempt to use this data, they do not properly display or update the extended characters.

This problem is quite complex. If any of your applications use the extended characters in the upper half of the code page, make sure that all clients and all machines using the database use the same or a compatible code page.

Indexes

All index types, except DATE, TIME, and DTTM, are supported for CHAR data and VARCHAR data less than or equal to 255 bytes in length.

VARCHAR data and trailing blanks

Data inserted using INSERT, UPDATE, or LOAD TABLE can be:

- Enclosed in quotes
- Not enclosed in quotes
- Binary

For a column of data type VARCHAR, trailing blanks within the data being inserted are handled as follows:

- For data enclosed in quotes, trailing blanks are never trimmed.
- For data not enclosed in quotes:
  - Trailing blanks always trimmed on insert and update.
  - For a LOAD statement, you can use the STRIP RTRIM/OFF LOAD option to specify whether to have the trailing blanks trimmed. The STRIP RTRIM/OFF option applies only to variable-length non-binary data. For example, assume the following schema:

```
CREATE TABLE t( c1 VARCHAR(3) );
LOAD TABLE t( c1 ',' ) ........ STRIP RTRIM    // trailing blanks trimmed

LOAD TABLE t( c1 ',' ) ........ STRIP OFF      // trailing blanks not trimmed
```

```
LOAD TABLE t( c1 ASCII(3) ) ... STRIP RTRIM    // trailing blanks not trimmed
LOAD TABLE t( c1 ASCII(3) ) ... STRIP OFF      // trailing blanks trimmed

LOAD TABLE t( c1 BINARY ) ..... STRIP RTRIM    // trailing blanks trimmed
LOAD TABLE t( c1 BINARY ) ..... STRIP OFF      // trailing blanks trimmed
```

- For binary data, trailing blanks are always trimmed.

When you write your applications, do not depend on the existence of trailing blanks in VARCHAR columns. If an application relies on trailing blanks, use a CHAR column instead of a VARCHAR column.

Restriction on CHAR and VARCHAR data over 255 bytes

Only the default index, WD, and CMP index types are supported for CHAR and VARCHAR columns over 255 bytes. You cannot create an LF, HG, HNG, DATE, TIME, or DTTM index for these columns.

Compatibility

- The CHARACTER (*n*) alternative for CHAR is not supported in Adaptive Server Enterprise.
- Sybase IQ does not support the NCHAR, NVARCHAR, UNICHAR, and UNIVARCHAR data types provided by Adaptive Server Enterprise. Sybase IQ supports Unicode in the CHAR and VARCHAR data types.
- Sybase IQ supports a longer LONG VARCHAR data type than SQL Anywhere. See *Large Objects Management in Sybase IQ*.
- For compatibility between Sybase IQ and Adaptive Server Enterprise, always specify a length for character data types.

Long strings

SQL Anywhere treats CHAR, VARCHAR, and LONG VARCHAR columns all as the same type. Values up to 254 characters are stored as short strings, with a preceding length byte. Any values that are longer than 255 bytes are considered long strings. Characters after the 255th are stored separate from the row containing the long string value.

There are several functions (see SQL Functions) that will ignore the part of any string past the 255th character. They are soundex, similar, and all of the date functions. Also, any arithmetic involving the conversion of a long string to a number will work on only the first 255 characters. It would be extremely unusual to run in to one of these limitations.

All other functions and all other operators work with the full length of long strings.

Numeric data types

Description

For storing numerical data.

Syntax

[**UNSIGNED**] **BIGINT**

[**UNSIGNED**] { **INT** | **INTEGER** }

SMALLINT

TINYINT

DECIMAL [(*precision* [, *scale*])]

NUMERIC [(*precision* [, *scale*])]

DOUBLE

FLOAT [(*precision*)]

REAL

Usage

BIGINT A signed 64-bit integer, requiring 8 bytes of storage.

You can specify integers as UNSIGNED. By default the data type is signed. Its range is between -9223372036854775808 and 9223372036854775807 (signed) or from 0 to 18446744073709551615 (unsigned).

INT *or* ***INTEGER*** A signed 32-bit integer with a range of values between -2147483648 and 2147483647 requiring 4 bytes of storage.

The INTEGER data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

You can specify integers as UNSIGNED; by default the data type is signed. The range of values for an unsigned integer is between 0 and 4294967295.

SMALLINT A signed 16-bit integer with a range between -32768 and 32767, requiring 2 bytes of storage.

The SMALLINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

TINYINT An unsigned 8-bit integer with a range between 0 and 255, requiring 1 byte of storage.

The TINYINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

DECIMAL A signed decimal number with *precision* total digits and with *scale* of the digits after the decimal point. The precision can equal 1 to 126, and the scale can equal 0 up to precision value. The defaults are scale = 38 and precision = 126. Results are calculated based on the actual data type of the column to ensure accuracy, but you can set the maximum scale of the result returned to the application. For more information, see the "MAX_CLIENT_NUMERIC_SCALE option" on page 420 and the SET OPTION statement in *Reference: Statements and Options*.

Table 3-2 lists the storage required for a decimal number.

Table 3-2: Storage size for a decimal number

Precision	Storage
1 to 4	2 bytes
5 to 9	4 bytes
10 to 18	8 bytes
19 to 126	See below

The storage requirement in bytes for a decimal value with a precision greater than 18 can be calculated using the following formula:

```
4 + 2 * (int(((prec - scale) + 3) / 4) +
int((scale + 3) / 4) + 1)
```

where *int* takes the integer portion of its argument. The storage used by a column is based upon the precision and scale of the column. Each cell in the column has enough space to hold the largest value of that precision and scale. For example:

```
NUMERIC(18,4) takes 8 bytes per cell
NUMERIC(19,4) takes 16 bytes per cell
```

The DECIMAL data type is an exact numeric data type; its accuracy is preserved to the least significant digit after arithmetic operations. Its maximum absolute value is the number of nines defined by [*precision* - *scale*], followed by the decimal point, and then followed by the number of nines defined by *scale*. The minimum absolute nonzero value is the decimal point, followed by the number of zeros defined by [*scale* - 1], then followed by a single one. For example:

```
NUMERIC (3,2) Max positive = 9.99 Min non-zero = 0.01
Max negative = -9.99
```

If neither precision nor scale is specified for the explicit conversion of NULL to NUMERIC, the default is NUMERIC(1,0). For example,

```
SELECT CAST( NULL AS NUMERIC ) A,
       CAST( NULL AS NUMERIC(15,2) ) B
```

is described as:

```
A NUMERIC(1,0)
B NUMERIC(15,2)
```

NUMERIC Same as DECIMAL.

DOUBLE A signed double-precision floating-point number stored in 8 bytes. The range of absolute, nonzero values is between 2.2250738585072014e-308 and 1.797693134862315708e+308. Values held as DOUBLE are accurate to 15 significant digits, but might be subject to rounding errors beyond the fifteenth digit.

The DOUBLE data type is an approximate numeric data type; it is subject to rounding errors after arithmetic operations.

FLOAT If *precision* is not supplied, the FLOAT data type is the same as the REAL data type. If *precision* supplied, then the FLOAT data type is the same as the REAL or DOUBLE data type, depending on the value of the precision. The cutoff between REAL and DOUBLE is platform-dependent, and it is the number of bits used in the mantissa of single-precision floating point number on the platform.

When a column is created using the FLOAT data type, columns on all platforms are guaranteed to hold the values to at least the specified minimum precision. In contrast, REAL and DOUBLE do not guarantee a platform-independent minimum precision.

The FLOAT data type is an approximate numeric data type; it is subject to rounding errors after arithmetic operations.

REAL A signed single-precision floating-point number stored in 4 bytes. The range of absolute, nonzero values is 1.175494351e-38 to 3.402823466e+38. Values held as REAL are accurate to 6 significant digits, but might be subject to rounding errors beyond the sixth digit.

The REAL data type is an approximate numeric data type; it is subject to rounding errors after arithmetic operations.

Notes

- The INTEGER, NUMERIC, and DECIMAL data types are sometimes called exact numeric data types, in contrast to the approximate numeric data types FLOAT, DOUBLE, and REAL. Only exact numeric data is guaranteed to be accurate to the least significant digit specified after arithmetic operations.

- Do not fetch TINYINT columns into Embedded SQL variables defined as CHAR or UNSIGNED CHAR, since the result is an attempt to convert the value of the column to a string and then assign the first byte to the variable in the program.

Indexes

- The CMP and HNG index types do not support the FLOAT, DOUBLE, and REAL data types, and the HG index type is not recommended.
- The WD, DATE, TIME, and DTTM index types do not support the numeric data types.

Compatibility

- In embedded SQL, fetch TINYINT columns into 2-byte or 4-byte integer columns. Also, to send a TINYINT value to a database, the C variable should be an integer.
- Adaptive Server Enterprise 12.5.x versions do not support unsigned integers. You can map Sybase IQ unsigned integers to Adaptive Server Enterprise signed integers or numeric data, and the data are converted implicitly.
  - Map IQ UNSIGNED SMALLINT data to ASE INT
  - If you have negative values, map IQ UNSIGNED BIGINT to ASE NUMERIC (*precision*, *scale*)

    To avoid performance issues for cross-database joins on UNSIGNED BIGINT columns, the best approach is to cast to a (signed) BIGINT on the Sybase IQ side.
- You should avoid default precision and scale settings for NUMERIC and DECIMAL data types, as these differ by product:

Database	Default precision	Default scale
Sybase IQ	126	38
Adaptive Server Enterprise	18	0
SQL Anywhere	30	6

- The FLOAT (*p*) data type is a synonym for REAL or DOUBLE, depending on the value of *p*. For Adaptive Server Enterprise, REAL is used for *p* less than or equal to 15, and DOUBLE for *p* greater than 15. For Sybase IQ, the cutoff is platform-dependent, but on all platforms, the cutoff value is greater than 22.

- Sybase IQ includes two user-defined data types, MONEY and SMALLMONEY, which are implemented as NUMERIC(19,4) and NUMERIC(10,4) respectively. They are provided primarily for compatibility with Adaptive Server Enterprise.

Binary data types

Description

For storing raw binary data, such as pictures, in a hexadecimal-like notation, up to a length of (32K – 1) bytes. The UNIQUEIDENTIFIER data type is used for storage of UUID (also known as GUID) values.

Syntax

BINARY [(*length*)]

VARBINARY [(*max-length*)]

UNIQUEIDENTIFIER

Usage

Binary data begins with the characters “0x” or “0X” and can include any combination of digits and the uppercase and lowercase letters A through F. You can specify the column length in bytes, or use the default length of 1 byte. Each byte stores 2 hexadecimal digits. Even though the default length is 1 byte, Sybase recommends that you always specify an even number of characters for BINARY and VARBINARY column length. If you enter a value longer than the specified column length, Sybase IQ truncates the entry to the specified length without warning or error.

BINARY Binary data of length *length* bytes. If *length* is omitted, the default is 1 byte. The maximum size allowed is 255 bytes. Use the fixed-length binary type BINARY for data in which all entries are expected to be approximately equal in length. Because entries in BINARY columns are zero-padded to the column length *length*, they might require more storage space than entries in VARBINARY columns.

VARBINARY Binary data up to a length of *max-length* bytes. If *max-length* is omitted, the default is 1 byte. The maximum size allowed is (32K – 1) bytes. Use the variable-length binary type VARBINARY for data that is expected to vary greatly in length.

Notes

As a separately licensed option, Sybase IQ supports binary large object (BLOB) data with a length ranging from zero (0) to 512TB (terabytes) for an IQ page size of 128KB or 2PB (petabytes) for an IQ page size of 512KB. The maximum length is equal to 4GB multiplied by the database page size. See *Large Objects Management in Sybase IQ*.

For information on LONG BINARY and IMAGE data types, see "Binary data types" on page 603.

Treatment of trailing zeros

All BINARY columns are padded with zeros to the full width of the column. Trailing zeros are truncated in all VARBINARY columns.

The following example creates a table with all four variations of BINARY and VARBINARY data types defined with NULL and NOT NULL. The same data is inserted in all four columns and is padded or truncated according to the data type of the column.

```
CREATE TABLE zeros (bnot BINARY(5) NOT NULL,
        bnull BINARY(5) NULL,
        vbnot VARBINARY(5) NOT NULL,
        vbnull VARBINARY(5) NULL);
INSERT zeros VALUES (0x12345000, 0x12345000,
        0x12345000, 0x12345000);
INSERT zeros VALUES (0x123, 0x123, 0x123, 0x123);
INSERT zeros VALUES (0x0, 0x0, 0x0, 0x0);
INSERT zeros VALUES ('002710000000ae1b',
'002710000000ae1b', '002710000000ae1b',
'002710000000ae1b');
SELECT * FROM zeros;
```

bnot	bnull	vbnot	vbnull
0x1234500000	0x1234500000	0x12345000	0x12345000
0x0123000000	0x0123000000	0x0123	0x0123
0x0000000000	0x0000000000	0x00	0x00
0x3030323731	0x3030323731	0x3030323731	0x3030323731

Because each byte of storage holds 2 hexadecimal digits, Sybase IQ expects binary entries to consist of the characters "0x" followed by an even number of digits. When the "0x" is followed by an odd number of digits, Sybase IQ assumes that you omitted the leading 0 and adds it for you.

Input values "0x00" and "0x0" are stored as "0x00" in variable-length binary columns (VARBINARY). In fixed-length binary columns (BINARY), the value is padded with zeros to the full length of the field:

```
INSERT zeros VALUES (0x0, 0x0, 0x0, 0x0);
SELECT * FROM zeros
```

bnot	bnull	vbnot	vbnull
0x0000000000	0x0000000000	0x00	0x00

If the input value does not include “0x”, Sybase IQ assumes that the value is an ASCII value and converts it. For example:

```
CREATE TABLE sample (col_bin BINARY(8));
INSERT sample VALUES ('002710000000ae1b');
SELECT * FROM sample;
```

col_bin
0x3030323731303030

Note In the above example, ensure you set the string_rtruncation option to “off”.

When you select a BINARY value, you must specify the value with the padded zeros or use the CAST function. For example:

```
SELECT * FROM zeros WHERE bnot = 0x0123000000;
```

or :

```
SELECT * FROM zeros WHERE bnot = CAST(0x0123 as
binary(5));
```

Loading ASCII data from a flat file

Any ASCII data loaded from a flat file into a binary type column (BINARY or VARBINARY) is stored as nibbles. For example, if 0x1234 or 1234 is read from a flat file into a binary column, Sybase IQ stores the value as hexadecimal 1234. Sybase IQ ignores the “0x” prefix. If the input data contains any characters out of the range 0 – 9, a – f, and A – F, the data is rejected.

Storage size

Table 3-3 lists the storage size of binary data.

Table 3-3: Storage size of binary data

Data type	Column definition	Input data	Storage
VARBINARY	width of (32K – 1) bytes	(32K – 1) bytes binary	(32K – 1) bytes
VARBINARY	width of (32K– 1) bytes	(64K – 2) bytes ASCII	(32K – 1) bytes
BINARY	width of 255 bytes	255 bytes binary	255 bytes
BINARY	width of 255 bytes	510 bytes ASCII	255 bytes

Platform dependence The exact form in which you enter a particular value depends on the platform you are using. Therefore, calculations involving binary data might produce different results on different machines.

For platform-independent conversions between hexadecimal strings and integers, use the INTTOHEX and HEXTOINT functions rather than the platform-specific CONVERT function. For details, see the section "Data type conversion functions" on page 108.

String operators

The concatenation string operators || and + both support binary type data. Explicit conversion of binary operands to character data types is not necessary with the || operator. Explicit and implicit data conversion produce different results, however.

Restrictions on BINARY and VARBINARY data

The following restrictions apply to columns containing BINARY and VARBINARY data:

- You cannot use the aggregate functions SUM, AVG, STDDEV, or VARIANCE with the binary data types. The aggregate functions MIN, MAX, and COUNT *do* support the binary data types BINARY and VARBINARY.
- HNG, WD, DATE, TIME, and DTTM indexes do not support BINARY or VARBINARY data.
- Only the default index and CMP index types are supported for VARBINARY data greater than 255 bytes in length.
- Bit operations are supported on BINARY and VARBINARY data that is 8 bytes or less in length.

Compatibility

The treatment of trailing zeros in binary data differs between Sybase IQ, SQL Anywhere, and Adaptive Server Enterprise:

Table 3-4: Treatment of trailing zeros

Data type	Sybase IQ	SQL Anywhere	ASE
BINARY NOT NULL	Padded	Not padded	Padded
BINARY NULL	Padded	Not padded	Not padded
VARBINARY NOT NULL	Truncated, not padded	Truncated, not padded	Truncated, not padded
VARBINARY NULL	Truncated, not padded	Truncated, not padded	Truncated, not padded

Adaptive Server Enterprise, SQL Anywhere, and Sybase IQ all support the STRING_RTRUNCATION database option, which affects error message reporting when an INSERT or UPDATE string is truncated. For Transact-SQL compatible string comparisons, set the STRING_RTRUNCATION option to the same value in both databases.

You can also set the STRING_RTRUNCATION option ON when loading data into a table, to alert you that the data is too large to load into the field. The default value is ON.

Bit operations on binary type data are not supported by Adaptive Server Enterprise. SQL Anywhere only supports bit operations against the first four bytes of binary type data. Sybase IQ supports bit operations against the first eight bytes of binary type data.

UNIQUEIDENTIFIER Used for storage of UUID (also known as GUID) values. The UNIQUEIDENTIFIER data type is often used for a primary key or other unique column to hold UUID (Universally Unique Identifier) values that can be used to uniquely identify rows. The NEWID function generates UUID values in such a way that a value produced on one computer does not match a UUID produced on another computer. UNIQUEIDENTIFIER values generated using NEWID can therefore be used as keys in a synchronization environment.

For example, the following statement updates the table mytab and sets the value of the column uid_col to a unique identifier generated by the NEWID function, if the current value of the column is NULL.

```
UPDATE mytab
    SET uid_col = NEWID()
      WHERE uid_col IS NULL
```

If you execute the following statement,

```
SELECT NEWID()
```

the unique identifier is returned as a BINARY(16). For example, the value might be 0xd3749fe09cf446e399913bc6434f1f08. You can convert this string into a readable format using the UUIDTOSTR() function.

UUID values are also referred to as GUIDs (Globally Unique Identifier).

The STRTOUUID and UUIDTOSTR functions are used to convert values between UNIQUEIDENTIFIER and string representations.

UNIQUEIDENTIFIER values are stored and returned as BINARY(16).

Because UNIQUEIDENTIFIER values are large, using UNSIGNED BIGINT or UNSIGNED INT identity columns instead of UNIQUEIDENTIFIER is more efficient, if you do not need cross database unique identifiers.

Standards and compatibility for UNIQUEIDENTIFIER

- **SQL92** Vendor extension.
- **Sybase** Supported by SQL Anywhere. Not supported by Adaptive Server Enterprise.

- **Backwards compatibility** In databases created before Sybase IQ version 12.7, the STRTOUUID, UUIDTOSTR, and NEWID functions were supported through CIS functional compensation. In versions 15.1 and later, the STRTOUUID, UUIDTOSTR, and NEWID functions are native Sybase IQ functions.

See also

For more information related to UNIQUEIDENTIFIER:

- "NEWID function [Miscellaneous]" on page 208
- "UUIDTOSTR function [String]" on page 331
- "STRTOUUID function [String]" on page 263

Bit data type

Description

For storing Boolean values.

Data type	Values	Supported by
BIT	0 or 1	Sybase IQ and Enterprise

Usage

BIT stores only the values 0 or 1. Inserting any nonzero value into a BIT column stores a 1 in the column. Inserting any zero value into a BIT column stores a 0.

Only the default index type is supported for BIT data.

Compatibility

Adaptive Server Enterprise BIT datatypes only allow 0 or 1 values.

Date and time data types

Description

For storing dates and times.

Syntax

DATE

DATETIME

SMALLDATETIME

TIME

TIMESTAMP

Usage

DATE A calendar date, such as a year, month and day. The year can be from 0001 to 9999. The day must be a nonzero value, so that the minimum date is 0001-01-01. A DATE value requires 4 bytes of storage.

DATETIME A domain, implemented as TIMESTAMP. DATETIME is provided primarily for compatibility with Adaptive Server Enterprise. For an exception, see "Compatibility of string to datetime conversions" on page 90.

SMALLDATETIME A domain, implemented as TIMESTAMP. SMALLDATETIME is provided primarily for compatibility with Adaptive Server Enterprise. For an exception, see "Compatibility of string to datetime conversions" on page 90.

TIME Time of day, containing hour, minute, second, and fraction of a second. The fraction is stored to 6 decimal places. A TIME value requires 8 bytes of storage. (ODBC standards restrict TIME data type to an accuracy of seconds. For this reason, do not use TIME data types in WHERE clause comparisons that rely on a higher accuracy than seconds.)

TIMESTAMP Point in time, containing year, month, day, hour, minute, second, and fraction of a second. The fraction is stored to 6 decimal places. The day must be a nonzero value. A TIMESTAMP value requires 8 bytes of storage.

The valid range of the TIMESTAMP data type is from 0001-01-01 00:00:00.000000 to 9999-12-31 23:59:59.999999. The display of TIMESTAMP data outside the range of 1600-02-28 23:59:59 to 7911-01-01 00:00:00 might be incomplete, but the complete datetime value is stored in the database; you can see the complete value by first converting it to a character string. You can use the CAST() function to do this, as in the following example, which first creates a table with DATETIME and TIMESTAMP columns, then inserts values where the date is greater 7911-01-01.

```
create table mydates (id int, descript char(20),
  datetime_null datetime, timestamp_null timestamp);
insert into mydates values (1, 'example', '7911-12-30
  23:59:59','7911-12-30 06:03:44');
commit;
```

When you select without using CAST, hours and minutes are set to 00:00:

```
select * from mydates;
```

```
1, 'example', '7911-12-30 00:00:59.000', '7911-12-30
00:00:44.000'
```

When you select using cast, you see the complete timestamp:

```
select id, descript, cast(datetime_null as char(21)),
  cast(timestamp_null as char(21)) from mydates;
```

```
1, 'example', '7911-12-30 23:59:59.0', '7911-12-30
06:03:44.0'
```

Notes

The following index types are supported by date and time data:

- All date and time data types support the CMP, HG, HNG, and LF index types; the WD index type is not supported.
- DATE data supports the DATE index.
- TIME data supports the TIME index.
- DATETIME and TIMESTAMP data support the DTTM index.

Sending dates and times to the database

Description

You can send dates and times to the database in one of the following ways:

- Using any interface, as a string
- Using ODBC, as a TIMESTAMP structure
- Using Embedded SQL, as a SQLDATETIME structure

When you send a time to the database as a string (for the TIME data type) or as part of a string (for TIMESTAMP or DATE data types), hours, minutes, and seconds must be separated by colons in the format *hh*:*mm*:*ss*:*sss*, but can appear anywhere in the string. As an option, a period can separate the seconds from fractions of a second, as in *hh*:*mm*:*ss*.*sss*. The following are valid and unambiguous strings for specifying times:

```
21:35 -- 24 hour clock if no am or pm specified
10:00pm -- pm specified, so interpreted as 12 hour clock
10:00 -- 10:00am in the absence of pm
10:23:32.234 -- seconds and fractions of a
                    second included
```

When you send a date to the database as a string, conversion to a date is automatic. You can supply the string in one of two ways:

- As a string of format *yyyy/mm/dd* or *yyyy-mm-dd*, which is interpreted unambiguously by the database
- As a string interpreted according to the DATE_ORDER database option

Date format strings cannot contain any multibyte characters. Only single-byte characters are allowed in a date/time/datetime format string, even when the collation order of the database is a multibyte collation order like 932JPN.

Retrieving dates and times from the database

Description

You can retrieve dates and times from the database in one of the following ways:

- Using any interface, as a string
- Using ODBC, as a TIMESTAMP structure
- Using embedded SQL, as a SQLDATETIME structure

When a date or time is retrieved as a string, it is retrieved in the format specified by the database options DATE_FORMAT, TIME_FORMAT and TIMESTAMP_FORMAT. For descriptions of these options, see SET OPTION statement in *Reference: Statements and Options*.

For information on functions dealing with dates and times, see "Date and time data types" on page 82. The following operators are allowed on dates:

- **timestamp + integer** Add the specified number of days to a date or timestamp.
- **timestamp - integer** Subtract the specified number of days from a date or timestamp.
- **date - date** Compute the number of days between two dates or timestamps.
- **date + time** Create a timestamp combining the given date and time.

Comparing dates and times

Description

To compare a date to a string *as a string*, use the DATEFORMAT function or CAST function to convert the date to a string before comparing. For example:

```
DATEFORMAT(invoice_date,'yyyy/mm/dd') = '1992/05/23'
```

You can use any allowable date format for the DATEFORMAT string expression.

Date format strings must not contain any multibyte characters. Only single-byte characters are allowed in a date/time/datetime format string, even when the collation order of the database is a multibyte collation order like 932JPN.

If '?' represents a multibyte character, the following query fails:

```
SELECT DATEFORMAT ( StartDate, ‘yy?’) FROM Employees;
```

Instead, move the multibyte character outside of the date format string using the concatenation operator:

```
SELECT DATEFORMAT (StartDate, ‘yy’) + ‘?’ FROM
Employees;
```

Using unambiguous dates and times

Description

Dates in the format *yyyy/mm/dd* or *yyyy-mm-dd* are always recognized as dates regardless of the DATE_ORDER setting. You can use other characters as separators; for example, a question mark, a space character, or a comma. You should use this format in any context where different users might be employing different DATE_ORDER settings. For example, in stored procedures, use of the unambiguous date format prevents misinterpretation of dates according to the user's DATE_ORDER setting.

A string of the form *hh*:*mm*:*ss.sss* is also interpreted unambiguously as a time.

For combinations of dates and times, any unambiguous date and any unambiguous time yield an unambiguous date-time value. Also, the following form is an unambiguous date-time value:

```
YYYY-MM-DD HH.MM.SS.SSSSSS
```

You can use periods in the time only in combination with a date.

In other contexts, you can use a more flexible date format. Sybase IQ can interpret a wide range of strings as formats. The interpretation depends on the setting of the database option DATE_ORDER. The DATE_ORDER database option can have the value ‘MDY’, ‘YMD’, or ‘DMY’. See SET OPTION statement in *Reference: Statements and Options*. For example, to set the DATE_ORDER option to ‘DMY’ enter:

```
SET OPTION DATE_ORDER = 'DMY' ;
```

The default DATE_ORDER setting is ‘YMD’. The ODBC driver sets the DATE_ORDER option to ‘YMD’ whenever a connection is made.Use the SET OPTION statement to change the value.

The database option DATE_ORDER determines whether the string 10/11/12 is interpreted by the database as Oct 11 1912, Nov 12 1910, or Nov 10 1912. The year, month, and day of a date string should be separated by some character (for example "/", "-", or space) and appear in the order specified by the DATE_ORDER option.

You can supply the year as either 2 or 4 digits. The value of the option NEAREST_CENTURY affects the interpretation of 2-digit years: 2000 is added to values less than NEAREST_CENTURY, and 1900 is added to all other values. The default value of this option is 50. Thus, by default, 50 is interpreted as 1950, and 49 is interpreted as 2049. See the "NEAREST_CENTURY option [TSQL]" in *Reference: Statements and Options*.

The month can be the name or number of the month. The hours and minutes are separated by a colon, but can appear anywhere in the string.

Sybase recommends that you always specify the year using the 4-digit format.

With an appropriate setting of DATE_ORDER, the following strings are all valid dates:

```
99-05-23 21:35
99/5/23
1999/05/23
May 23 1999
23-May-1999
Tuesday May 23, 1999 10:00pm
```

If a string contains only a partial date specification, default values are used to fill out the date. The following defaults are used:

year 1900

month No default

day 1 (useful for month fields; for example, 'May 1999' is the date '1999-05-01 00:00')

hour, minute, second, fraction 0

Domains

Description

Domains are aliases for built-in data types, including precision and scale values where applicable.

Domains, also called user-defined data types, allow columns throughout a database to be defined automatically on the same data type, with the same NULL or NOT NULL condition. This encourages consistency throughout the database. Domain names are case-insensitive. Sybase IQ returns an error if you attempt to create a domain with the same name as an existing domain except for case.

Simple domains

You create domains using the CREATE DOMAIN statement. See “CREATE DOMAIN statement” in *Reference: Statements and Options*.

The following statement creates a data type named street_address, which is a 35-character string:

```
CREATE DOMAIN street_address CHAR( 35 )
```

You can use CREATE DATATYPE as an alternative to CREATE DOMAIN, but this is not recommended, as CREATE DOMAIN is the syntax used in the draft SQL/3 standard.

Resource authority is required to create data types. Once a data type is created, the user ID that executed the CREATE DOMAIN statement is the owner of that data type. Any user can use the data type, and unlike other database objects, the owner name is never used to prefix the data type name.

The street_address data type may be used in exactly the same way as any other data type when defining columns. For example, the following table with two columns has the second column as a street_address column:

```
CREATE TABLE twocol (id INT,
street street_address)
```

Owners or DBAs can drop domains by issuing a COMMIT and then using the DROP DOMAIN statement:

```
DROP DOMAIN street_address
```

You can carry out this statement only if no tables in the database are using data type.

Constraints and defaults with user-defined data types

Many of the attributes associated with columns, such as allowing NULL values, having a DEFAULT value, and so on, can be built into a user-defined data type. Any column that is defined on the data type automatically inherits the NULL setting, CHECK condition, and DEFAULT values. This allows uniformity to be built into columns with a similar meaning throughout a database.

For example, many primary key columns in the sample database are integer columns holding ID numbers. The following statement creates a data type that may be useful for such columns:

```
CREATE DOMAIN id INT
NOT NULL
DEFAULT AUTOINCREMENT
CHECK( @col > 0 )
```

Any column created using the data type id is not allowed to hold NULLs, defaults to an autoincremented value, and must hold a positive number. Any identifier could be used instead of *col* in the *@col* variable.

The attributes of the data type can be overridden if needed by explicitly providing attributes for the column. A column created on data type id with NULL values explicitly allowed does allow NULLs, regardless of the setting in the id data type.

Compatibility

- **Named constraints and defaults** In Sybase IQ, user-defined data types are created with a base data type, and optionally, a NULL or NOT NULL condition. Named constraints and named defaults are not supported.
- **Creating data types** In Sybase IQ, you can use the sp_addtype system procedure to add a domain, or you can use the CREATE DOMAIN statement. In Adaptive Server Enterprise, you must use sp_addtype.

Data type conversions

Description

Type conversions happen automatically, or you can explicitly request them using the CAST or CONVERT function.

If a string is used in a numeric expression or as an argument to a function expecting a numeric argument, the string is converted to a number before use.

If a number is used in a string expression or as a string function argument, then the number is converted to a string before use.

All date constants are specified as strings. The string is automatically converted to a date before use.

There are certain cases where the automatic data type conversions are not appropriate.

```
'12/31/90' + 5 -- Tries to convert the string to a number
'a' > 0        -- Tries to convert 'a' to a number
```

You can use the CAST or CONVERT function to force type conversions.

The following functions can also be used to force type conversions:

- DATE(expression) – converts the expression into a date, and removes any hours, minutes or seconds. Conversion errors might be reported.
- DATETIME(expression) – converts the expression into a timestamp. Conversion errors might be reported.
- STRING(expression) – similar to `CAST(value AS CHAR)`, except that `string(NULL)` is the empty string (''), whereas `CAST(NULL AS CHAR)` is the NULL value.

For information about the CAST and CONVERT functions, see “Data type conversion functions” on page 108.

Compatibility of string to datetime conversions

There are some differences in behavior between Sybase IQ and Adaptive Server Enterprise when converting strings to date and time data types.

If you convert a string containing only a time value (no date) to a date/time data type, Sybase IQ and Adaptive Server Enterprise both use a default date of January 1, 1900. SQL Anywhere uses the current date.

If the milliseconds portion of a time is less than 3 digits, Adaptive Server Enterprise interprets the value differently depending on whether it was preceded by a period or a colon. If preceded by a colon, the value means thousandths of a second. If preceded by a period, 1 digit means tenths, 2 digits mean hundredths, and 3 digits mean thousandths. Sybase IQ and SQL Anywhere interpret the value the same way, regardless of the separator.

Example

- Adaptive Server Enterprise converts the values below as shown.

```
12:34:56.7 to 12:34:56.700
12.34.56.78 to 12:34:56.780
12:34:56.789 to 12:34:56.789
12:34:56:7 to 12:34:56.007
12.34.56:78 to 12:34:56.078
12:34:56:789 to 12:34:56.789
```

- Sybase IQ converts the milliseconds value in the manner that Adaptive Server Enterprise does for values preceded by a period, in both cases:

```
12:34:56.7 to 12:34:56.700
12.34.56.78 to 12:34:56.780
12:34:56.789 to 12:34:56.789
12:34:56:7 to 12:34:56.700
12.34.56:78 to 12:34:56.780
12:34:56:789 to 12:34:56.789
```

Compatibility of exported dates

For dates in the first 9 days of a month and hours less than 10, Adaptive Server Enterprise supports a blank for the first digit; Sybase IQ supports a zero or a blank. For details on how to load such data from Adaptive Server Enterprise into Sybase IQ, see Chapter 7, “Moving Data In and Out of Databases” in *System Administration Guide: Volume 1*.

Conversion of BIT to BINARY data type

Sybase IQ supports BIT to BINARY and BIT to VARBINARY implicit and explicit conversion and is compatible with Adaptive Server Enterprise support of these conversions. Sybase IQ implicitly converts BIT to BINARY and BIT to VARBINARY data types for comparison operators, arithmetic operations, and INSERT and UPDATE statements.

For BIT to BINARY conversion, bit value ‘b’ is copied to the first byte of the binary string and the rest of the bytes are filled with zeros. For example, BIT value 1 is converted to BINARY(n) string 0x0100...00 having 2^n nibbles. BIT value 0 is converted to BINARY string 0x00...00.

For BIT to VARBINARY conversion, BIT value ‘b’ is copied to the first byte of the BINARY string and the remaining bytes are not used; that is, only one byte is used. For example, BIT value 1 is converted to VARBINARY(n) string 0x01 having 2 nibbles.

The result of both implicit and explicit conversions of BIT to BINARY and BIT to VARBINARY data types is the same. The following table contains examples of BIT to BINARY and VARBINARY conversions.

Conversion of BIT value ‘1’ to	Result
BINARY(3)	0x010000
VARBINARY(3)	0x01
BINARY(8)	0x0100000000000000
VARBINARY(8)	0x01

BIT to BINARY and BIT to VARBINARY conversion examples These examples illustrate both implicit and explicit conversion of BIT to BINARY and BIT to VARBINARY data types.

Given the following tables and data:

```
CREATE TABLE tbin(c1 BINARY(9))
CREATE TABLE tvarbin(c2 VARBINARY(9))
CREATE TABLE tbar(c2 BIT)

INSERT tbar VALUES(1)
INSERT tbar VALUES(0)
```

Implicit conversion of BIT to BINARY:

```
INSERT tbin SELECT c2 FROM tbar

c1
---
0x010000000000000000   (18 nibbles)
0x000000000000000000   (18 nibbles)
```

Implicit conversion of BIT to VARBINARY:

```
INSERT tvarbin SELECT c2 FROM tbar

c2
---
0x01
0x00
```

Explicit conversion of BIT to BINARY:

```
INSERT tbin SELECT CONVERT (BINARY(9), c2) FROM tbar

c1
---
0x010000000000000000   (18 nibbles)
0x000000000000000000   (18 nibbles)
```

Explicit conversion of BIT to VARBINARY:

```
INSERT tvarbin SELECT CONVERT(VARBINARY(9), c2) FROM
tbar

c2
---
0x01
0x00
```

Conversion between BIT and CHAR/VARCHAR data types

Sybase IQ supports implicit conversion between BIT and CHAR, and BIT and VARCHAR data types for comparison operators, arithmetic operations, and INSERT and UPDATE statements.

BIT to VARCHAR, CHAR to BIT, and VARCHAR to BIT conversion examples These examples illustrate both implicit and explicit conversions between BIT and CHAR, and BIT and VARCHAR data types.

Given the following tables and data:

```
CREATE TABLE tchar(c1 CHAR(9))
CREATE TABLE tvarchar(c2 VARCHAR(9))
CREATE TABLE tbar(c2 BIT)
CREATE TABLE tbit(c2 BIT)
```

```
INSERT tbar VALUES(1)
INSERT tbar VALUES(0)
```

Implicit conversion of BIT to VARCHAR / VARCHAR to BIT and implicit conversion of BIT to VARCHAR:

```
INSERT tvarchar SELECT c2 FROM tbar
SELECT c2, char_length(c2) FROM tvarchar

c2,char_length(tvarchar.c2)
---------------------------
‘1’,1
‘0’,1
```

Implicit conversion of VARCHAR to BIT:

```
INSERT tbit SELECT c2 FROM tvarchar
SELECT c2 FROM tbit

c2
--
0
1
```

Explicit conversion of BIT to CHAR / CHAR to BIT and explicit conversion of BIT to CHAR:

```
INSERT tchar SELECT CONVERT (CHAR(9), c2) FROM tbar
SELECT c1, char_length(c1) FROM tchar

c1,char_length(tchar.c1)
------------------------
‘1’,9
‘0’,9
```

Explicit conversion of CHAR to BIT:

```
INSERT tbit SELECT CONVERT (BIT, c1) FROM tchar
SELECT c2 FROM tbit

c2
--
0
1
```

Explicit conversion of BIT to VARCHAR / VARCHAR to BIT and explicit conversion of BIT to VARCHAR:

```
INSERT tvarchar SELECT CONVERT(VARCHAR(9), c2)
  FROM tbar
```

```
SELECT c2, char_length(c2) FROM tvarchar

c2,char_length(tvarchar.c2)
---------------------------
‘1’,1
‘0’,1
```

Explicit conversion of VARCHAR to BIT:

```
INSERT tbit SELECT CONVERT (BIT, c2) FROM tvarchar
SELECT c2 FROM tbit

c2
--
0
1
```

Sybase IQ binary load format

Description

Sybase IQ uses the FORMAT BINARY and BINARY column specification clauses to produce data files that can be read by the LOAD TABLE statement.

To speed data loading into Sybase IQ, customers can create data files in Sybase IQ binary format and load this data into Sybase IQ using the FORMAT BINARY syntax of the LOAD TABLE command.

You can find instructions for creating a load script using the LOAD TABLE syntax and specifying the load specification in *Reference: Statements and Options*.

Create data files with these binary formats to load into columns with the corresponding data types. In most cases, Sybase IQ uses the platform-specific binary format. These data types are exceptions that use binary formats that are specific to Sybase IQ:

- DATE
- TIME
- DATETIME
- NUMERIC

IQ binary load format and load efficiency

The Sybase IQ binary load format is a fixed width format.

In general, fixed width loads complete faster than variable width loads. When the load logic knows the length of a column and row, the data is processed more efficiently. Using delimiters to separate columns and rows that vary in width causes the load to spend time scanning the input data looking for them.

The IQ Binary Load Format is a fixed width load. The load can determine the width of each column and length of each row from information in the table definition.

Note Binary load format is endian-sensitive. This is because binary load format utilizes native binary data types to represent data.

Operating system native data types

Data for the following data types is stored in native operating system binary format and can be written to data files directly in that format. Sybase IQ reads the respective number of bytes directly into the associated data types without conversion.

- BIT (1 byte)
- TINYINT (1 byte)
- SMALLINT (2 bytes)
- INT/UNSIGNED INT (4 bytes)
- BIGINT/UNSIGNED BIGINT (8 bytes)
- FLOAT (4 bytes)
- DOUBLE (8 bytes)
- CHAR/VARCHAR (character data)
- BINARY/VARBINARY (binary data)

By default, VARCHAR and VARBINARY columns are read in as many bytes as specified by LOAD TABLE *column-spec*.

DATE

DATE column data is stored in Sybase IQ as four bytes (a 32-bit unsigned integer) representing the number of days since 0000-01-01. To convert a calendar date to the Sybase IQ binary format, use:

For a given year, month, and day:

```
year = current_year - 1;
days_in_year_0000 = 366;
binaryDateValue = (year * 365)
+ (year / 4)
- (year / 100)
```

```
+ (year / 400)
days_in_year_0000
day_of_current_year
-1;
```

For the day_of_current_year value in the formula above, consider the following example: February 12 is day 43.

TIME

TIME data is stored as a 64-bit unsigned quantity that represents a number in microseconds (in other words, 1.0e-6 seconds). The microsecond quantity is computed using:

For a given hour, minute, second, and microsecond (usec):

```
binaryTimeValue = (hour * 3600 + minute * 60 + second +
microsecond ) * 1000000
```

TIMESTAMP

TIMESTAMP data is stored as a 64-bit unsigned integer and represents a quantity in microseconds. You can compute a binary TIMESTAMP value using:

For a given year, month, day, hour, minute, second, and microsecond:

Compute *binaryDateValue* for the date as shown above.
Compute *binaryTimeValue* for the time as shown above.

```
binaryDateTimeValue = binaryDateValue *
    86400000000 + binaryTimeValue
```

NUMERIC and DECIMAL

Formats for NUMERIC and DECIMAL data types vary as a function of precision. The value must be right-padded with zeroes to the full scale of the value. The value must also be fully left-padded with zeroes, but padding happens automatically with binary programming. Once the values are padded, the decimal point is removed. For example, the value 12.34 looks like:

- NUMERIC(4,2): 1234
- NUMERIC(6,4): 123400
- NUMERIC(8,4): 00123400
- NUMERIC(12,6): 000012340000
- NUMERIC(16,8): 0000001234000000

After the value is padded and the decimal point is removed, the following rules apply:

- If precision <= 4, then the binary format is identical to native operating system binary format for 2 byte integer quantity.
- If precision is between 5 and 9, then the binary format is identical to native operating system binary format for a 4 byte integer quantity.

- If precision is between 10 and 18, then the binary format is identical to native operating system binary format for an 8 byte integer quantity.
- If precision >= 19, then there is a special format that uses the following C++ struct definition:

```
struct {
unsigned char sign; // sign 1 for +, 0 for -
unsigned char ndig; // # digits
unsigned char exp; // exponent
unsigned short digits[80];
};
```

Exponent is excess-80 form, unless the value is zero. A “zero” value is represented as:

```
sign = 1
ndig = 0
exp = 0
```

The maximum exponent value is 159. The maximum number of supported digits is 288. “digits[0]” contains the least-significant digits. Digits are stored in a packed representation with two digits per “unsigned short” (2-byte) quantity. For a given “digit:”

```
lower order digit = digit[i] & 0x00FF

high order digit => digit[i] & 0xFF00
```

For example, consider the value 100 loaded into a NUMERIC(20) column. The binary layout of this value is:

```
0x0101 0x5000 0x0064 0x0000 0x0000 ......

Sign - 0x01
Number digits - 0x01
Exponent - 0x50
Digits - 0x0064
```

As another example, consider the value 32769:

```
0x0102 0x5000 0x0ad1 0x0003 0x0000 0x0000 ....

Sign - 0x01
Number digits - 0x02
Exponent - 0x50
Digits - 0x0ad1 0x0003
```

If you translate the digits into base 10, you have:

```
0x0ad1 => 2769  0x0003 => 3
```

Inserting NULL

The most expedient way to insert NULL values is to use the NULL BYTE in the input file and specify WITH NULL BYTE in the column specification of the LOAD TABLE statement. This is done by terminating each data field in the input file with 'x00' or 'x01'. Terminating a data field in the input file with 'x01' instructs the load to insert NULL into the column. For example:

```
create table d1 ( c1 date );
load table d1 ( c1 binary with null byte ) from
'filename' quotes off escapes off format binary;
```

If the content of the load input file is 000b32cb00000b32cc00, two rows will be loaded to the table. The first row will be May 7, 2009 and the second May 8, 2009. Notice that a NULL BYTE was added to the input file after each binary date. If you want NULL loaded into the first row, change the value of the NULL BYTE in the input file to 'x01'.

```
000b32cb01000b32cc00
```

The NULL portion of the column specification indicates how to treat certain input values as NULL values, when loading into the table column. These characters can include BLANKS, ZEROS, or any other list of literals you define. When you specify a NULL value or read a NULL value from the source file, the destination column must be able to contain NULLs.

ZEROS is interpreted as follows:

- The column is set to NULL if the input data is all binary zeros (not character zeros).
- If the input data is character zero:
  - NULL(ZEROS) never causes the column to be NULL.
  - NULL('0') causes the column to be NULL. For example:

    Load:

    ```
    CREATE TABLE t1 ( c1 INT, c2 INT );
    ```

    View the input data file, which uses big-endian byte ordering:

    ```
    od -x data.inp
    3030 3030 0000 04d2
    ```

    Execute:

    ```
    LOAD TABLE t1 ( c1 ASCII(4) NULL( '0000' ),
        c2 BINARY )
    FROM 'data.inp'
        FORMAT BINARY
        QUOTES OFF
    ```

```
ESCAPES OFF;
```

The results:

```
SELECT * FROM t1;
c1      c2
NULL    1234
```

- If the input data is binary zero (all bits clear):
  - NULL(ZEROS) causes the column to be NULL.
  - NULL('0') never causes the column to be NULL, for example:

  Load:

  ```
  CREATE TABLE t1 ( c1 INT, C2 INT );
  ```

  VIEW the input data file, which uses big-endian byte ordering:

  ```
  od -x data.inp
  0000 0000 0000 04d2
  ```

  Execute:

  ```
  LOAD TABLE t1 ( c1 ASCII(4) NULL( zeros ),
      c2 BINARY )
  FROM 'data.inp'
      FORMAT BINARY
      QUOTES OFF
      ESCAPES OFF;
  ```

  The results:

  ```
  SELECT * FROM T1;
  c1      c2
  NULL    1234
  ```

  As another example, if your LOAD TABLE statement includes `col1 date('yymmdd') null(zeros)` and the data to load is 000000, you receive an error indicating that 000000 cannot be converted to a DATE(4). To get LOAD TABLE to insert a NULL value in col1 when the data is 000000, either write the NULL clause as `null('000000')`, or modify the data to equal binary zeros and use NULL(ZEROS).

Another way to load NULLs during a binary load is not to supply data for the column in the LOAD TABLE statement, if the destination column accepts null values. For example:

```
CREATE TABLE t1 ( c1 INT, c2 INT );
LOAD TABLE T1 ( c2 BINARY ) FROM 'data.inp'
    FORMAT BINARY
```

```
    QUOTES OFF
    ESCAPES OFF;

SELECT * FROM T1;
c1      c2
NULL    1234
Null    1234
```

View the input data file, which uses big-endian byte ordering:

```
od -x data.inp
0000 04d2 0000 04d2
```

CHAPTER 4 SQL Functions

About this chapter

This chapter describes the built-in functions that Sybase IQ supports.

Contents

Overview

Functions return information from the database and are allowed anywhere an expression is allowed.

When using functions with Sybase IQ:

- Unless otherwise stated, any function that receives the NULL value as a parameter returns a NULL value.
- If you omit the FROM clause, or if all tables in the query are in the SYSTEM dbspace, SQL Anywhere processes the query, instead of Sybase IQ, and might behave differently, especially with regard to syntactic and semantic restrictions and the effects of option settings. See the SQL Anywhere documentation for rules that might apply to processing.
- If you have a query that does not require a FROM clause, you can force Sybase IQ to process the query by adding the clause “FROM iq_dummy,” where iq_dummy is a one-row, one-column table that you create in your database.

Aggregate functions

Function

Aggregate functions summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement.

Usage

Simple aggregate functions, such as SUM(), MIN(), MAX(), AVG() and COUNT() are allowed only in the select list and in the HAVING and ORDER BY clauses of a SELECT statement. These functions summarize data over a group of rows from the database. Groups are formed using the GROUP BY clause of the SELECT statement.

A new class of aggregate functions, called **window functions**, provides moving averages and cumulative measures that compute answers to queries such as, “What is the quarterly moving average of the Dow Jones Industrial average,” or “List all employees and their cumulative salaries for each department.”

- Simple aggregate functions, such as AVG(), COUNT(), MAX(), MIN(), and SUM() summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement.

- Newer statistical aggregate functions that take one argument include STDDEV(), STDDEV_SAMP(), STDDEV_POP(), VARIANCE(), VAR_SAMP(), and VAR_POP().

Both the simple and newer categories of aggregates can be used as a windowing function that incorporates a <WINDOW CLAUSE> in a SQL query specification (a **window**) that conceptually creates a moving window over a result set as it is processed. See “Analytical functions” on page 104.

Another class of window aggregate functions supports analysis of time series data. Like the simple aggregate and statistical aggregate functions, you can use these window aggregates in conjunction with a SQL query specification (or *window-spec*). The time series window aggregate functions calculate correlation, linear regression, ranking, and weighted average results:

- ANSI SQL:2008 OLAP functions for time series analysis include: CORR(), COVAR_POP(), COVAR_SAMP(), CUME_DIST(), FIRST_VALUE(), LAST_VALUE(), REGR_AVGX(), REGR_AVGY(), REGR_COUNT(), REGR_INTERCEPT(), REGR_R2(), REGR_SLOPE(), REGR_SXX(), REGR_SXY(), and REGR_SYY().
- Non-ANSI SQL:2008 OLAP aggregate function extensions used in the database industry include FIRST_VALUE(), MEDIAN(), and LAST_VALUE().
- Weighted OLAP aggregate functions that calculate weighted moving averages include EXP_WEIGHTED_AVG() and WEIGHTED_AVG().

Table 4-1: Aggregate functions

Aggregate function	**Parameters**
AVG	([DISTINCT] { *column-name* | *numeric-expr* })
CORR	(*dependent-expression*, *independent-expression*)
COUNT	(*)
COUNT	([DISTINCT] { *column-name* | *numeric-expr* })
COVAR_POP	(*dependent-expression*, *independent-expression*)
COVAR_SAMP	(*dependent-expression*, *independent-expression*)
CUME_DIST	( )
EXP_WEIGHTED_AVG	(*expression*, *period-expression*)
FIRST_VALUE	(*expression*)
LAST_VALUE	(*expression*)
MAX	([DISTINCT] { *column-name* | *numeric-expr* })
MEDIAN	(*expression*)
MIN	([DISTINCT] { *column-name* | *numeric-expr* })
REGR_AVGX	(*dependent-expression*, *independent-expression*)

Aggregate function	**Parameters**
REGR_AVGY	(*dependent-expression*, *independent-expression*)
REGR_COUNT	(*dependent-expression*, *independent-expression*)
REGR_INTERCEPT	(*dependent-expression*, *independent-expression*)
REGR_R2	(*dependent-expression*, *independent-expression*)
REGR_SLOPE	(*dependent-expression*, *independent-expression*)
REGR_SXX	(*dependent-expression*, *independent-expression*)
REGR_SXY	(*dependent-expression*, *independent-expression*)
REGR_SYY	(*dependent-expression*, *independent-expression*)
STDDEV	([ALL] *expression*)
SUM	([DISTINCT] { *column-name* | *numeric-expr* })
VARIANCE	([ALL] *expression*)
WEIGHTED_AVG	(*expression*, *period-expression*

The aggregate functions AVG, SUM, STDDEV, and VARIANCE do not support the binary data types (BINARY and VARBINARY).

See also

See the individual analytical function descriptions in this chapter for specific details on the use of each function.

For more information about using OLAP functions, see Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2.*

Analytical functions

Function

Analytical functions include:

- Simple aggregates — AVG, COUNT, MAX, MIN, and SUM, STDDEV, and VARIANCE

  Note You can use all simple aggregates, except the Grouping() function, with an OLAP windowed function.

- Window functions:
  - Windowing aggregates — AVG, COUNT, MAX, MIN, and SUM.
  - Ranking functions — RANK, DENSE_RANK, PERCENT_RANK, and NTILE.

    - Statistical functions — STDDEV, STDDEV_SAMP, STDDEV_POP, VARIANCE, VAR_SAMP, and VAR_POP.
    - Distribution functions — PERCENTILE_CONT and PERCENTILE_DISC.
- Numeric functions — WIDTH_BUCKET, CEIL, and LN, EXP, POWER, SQRT, and FLOOR.

Windowing aggregate function usage

A major feature of the ANSI SQL extensions for OLAP is a construct called a **window**. This windowing extension let users divide result sets of a query (or a logical partition of a query) into groups of rows called partitions and determine subsets of rows to aggregate with respect to the current row.

You can use three classes of window functions with a window: ranking functions, the row numbering function, and window aggregate functions.

Windowing extensions specify a window function type over a window name or specification and are applied to partitioned result sets within the scope of a single query expression. A window partition is a subset of rows returned by a query, as defined by one or more columns in a special OVER clause:

```
OVER (PARTITION BY col1, col2...)
```

Windowing operations let you establish information such as the ranking of each row within its partition, the distribution of values in rows within a partition, and similar operations. Windowing also lets you compute moving averages and sums on your data, enhancing the ability to evaluate your data and its impact on your operations.

A window partition is a subset of rows returned by a query, as defined by one or more columns in a special OVER() clause:

```
OVER (PARTITION BY col1, col2...)
```

Ranking functions usage

The OLAP ranking functions let application developers compose single-statement SQL queries that answer questions such as “Name the top 10 products shipped this year by total sales,” or “Give the top 5% of salespeople who sold orders to at least 15 different companies.” These functions include the ranking functions, RANK(), DENSE_RANK(), PERCENT_RANK() and NTILE() with a PARTITION BY clause.

Rank analytical functions rank items in a group, compute distribution, and divide a result set into a number of groupings. The rank analytical functions, RANK, DENSE_RANK, PERCENT_RANK, and NTILE all require an OVER (ORDER BY) clause. For example:

```
RANK() OVER ( [PARTITION BY] ORDER BY <expression>
[ ASC | DESC ] )
```

The ORDER BY clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the OVER clause and is *not* an ORDER BY for SELECT. No aggregation functions in the rank query are allowed to specify DISTINCT.

The OVER clause indicates that the function operates on a query result set. The result set is the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses have all been evaluated. The OVER clause defines the data set of the rows to include in the computation of the rank analytical function.

The value *expression* is a sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

The ASC or DESC parameter specifies the ordering sequence as ascending or descending. Ascending order is the default.

Rank analytical functions are only allowed in the select list of a SELECT or INSERT statement or in the ORDER BY clause of the SELECT statement. Rank functions can be in a view or a union. You cannot use rank functions in a subquery, a HAVING clause, or in the select list of an UPDATE or DELETE statement. More than one rank analytical function is allowed per query in Sybase IQ 15.1.

Statistical aggregate analytic function usage

Summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement. Aggregate functions are allowed only in the select list and in the HAVING and ORDER BY clauses of a SELECT statement. These functions include STDDEV, STDDEV_POP, STDDEV_SAMP, VARIANCE, VAR_POP, and VAR_SAMP.

The OLAP functions can be used as a window function with an OVER() clause in a SQL query specification that conceptually creates a moving window over a result set as it is processed.

Distribution functions usage

The inverse distribution analytical functions PERCENTILE_CONT and PERCENTILE_DISC take a percentile value as the function argument and operate on a group of data specified in the WITHIN GROUP clause, or operate on the entire data set. These functions return one value per group. For PERCENTILE_DISC, the data type of the results is the same as the data type of its ORDER BY item specified in the WITHIN GROUP clause. For PERCENTILE_CONT, the data type of the results is either numeric, if the ORDER BY item in the WITHIN GROUP clause is a numeric, or double, if the ORDER BY item is an integer or floating-point.

The inverse distribution analytical functions require a WITHIN GROUP (ORDER BY) clause. For example:

```
PERCENTILE_CONT ( expression1 ) WITHIN GROUP ( ORDER BY
expression2 [ASC | DESC ] )
```

The value of *expression1* must be a constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, then a “wrong argument for percentile” error is returned. If the argument value is less than 0, or greater than 1, then a “data value out of range” error is returned.

The ORDER BY clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the WITHIN GROUP clause and is *not* an ORDER BY for the SELECT.

The WITHIN GROUP clause distributes the query result into an ordered data set from which the function calculates a result.

The value *expression2* is a sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

The ASC or DESC parameter specifies the ordering sequence as ascending or descending. Ascending order is the default.

Inverse distribution analytical functions are allowed in a subquery, a HAVING clause, a view, or a union. The inverse distribution functions can be used anywhere the simple non analytical aggregate functions are used. The inverse distribution functions ignore the NULL value in the data set.

Table 4-2 lists the analytical functions and their parameters. Unlike aggregate functions in Table 4-1, you cannot specify DISTINCT in window functions.

Table 4-2: Analytical functions

Function	Parameters
AVG	({ *column-name* | *numeric-expr* })
COUNT	(*)
COUNT	({ *column-name* | *expression* })
DENSE_RANK	()
GROUPING *	({ GROUPING *group-by-expression* })
MAX	({ *column-name* | *expression* })
MIN	({ *column-name* | *expression* })
NTILE	(*integer*)
PERCENT_RANK	()
PERCENTILE_CONT	(*numeric-expr*)
PERCENTILE_DISC	(*numeric-expr*)

Function	Parameters
RANK	()
STDDEV	([ALL] *expression*)
STDDEV_POP	([ALL] *expression*)
STDDEV_SAMP	([ALL] *expression*)
SUM	({ *column-name* | *expression* })
VAR_POP	([ALL] *expression*)
VAR_SAMP	([ALL] *expression*)
VARIANCE	([ALL] *expression*)

* The OLAP SQL standard allows Grouping() in GROUP BY CUBE, or GROUP BY ROLLUP operations only.

Compatibility

The ranking and inverse distribution analytical functions are not supported by Adaptive Server Enterprise.

See also

See the individual analytical function descriptions in this chapter for specific details on the use of each function.

See Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

Data type conversion functions

Function

Data type conversion functions convert arguments from one data type to another.

Table 4-3 lists the data type conversion functions and their parameters.

Table 4-3: Date type conversion functions

Data type conversion function	Parameters
BIGINTTOHEX	(*integer-expression*)
CAST	(*expression* AS *data type*)
CONVERT	(*data type*, *expression* [, *format-style*])
HEXTOBIGINT	(*hexadecimal-string*)
HEXTOINT	(*hexadecimal-string*)
INTTOHEX	(*integer-expr*)
ISDATE	(*string*)
ISNUMERIC	(*string*)

Description

The DATE, DATETIME, DATEFORMAT, and YMD functions that convert expressions to dates, timestamps, or strings based on a date format are listed in "Date and time functions" on page 109. The STRING function, which converts expressions to a string, is discussed in the section "String functions" on page 115.

The database server carries out many type conversions automatically. For example, if a string is supplied where a numerical expression is required, the string is automatically converted to a number. For more information on automatic data type conversions carried out by Sybase IQ, see "Data type conversions" on page 89.

Date and time functions

Function

Date and time functions perform conversion, extraction, or manipulation operations on date and time data types and can return date and time information.

Table 4-4 and Table 4-5 list the date and time functions and their parameters.

Syntax 1

Table 4-4: Date and time functions

Date and time functions	Parameters
DATE	(*expression*)
DATECEILING	(*date-part*, *datetime-expr*, *[multiple-expr]*)
DATEFLOOR	(*date-part*, *datetime-expr*, *[multiple-expr]*)
DATEFORMAT	(*datetime-expr*, *string-expr*)
DATENAME	(*date-part*, *date-expr*)
DATEROUND	(*date-part*, *datetime-expr*, *[multiple-expr]*)
DATETIME	(*expression*)
DAY	(*date-expr*)
DAYNAME	(*date-expr*)
DAYS	(*date-expr*)
DAYS	(*date-expr*, *date-expr*)
DAYS	(*date-expr*, *integer-expr*)
DOW	(*date-expr*)
HOUR	(*datetime-expr*)
HOURS	(*datetime-expr*)
HOURS	(*datetime-expr*, *datetime-expr*)
HOURS	(*datetime-expr*, *integer-expr*)
ISDATE	(*string*)
MINUTE	(*datetime-expr*)
MINUTES	(*datetime-expr*)
MINUTES	(*datetime-expr*, *datetime-expr*)
MINUTES	(*datetime-expr*, *integer-expr*)
MONTH	(*date-expr*)
MONTHNAME	(*date-expr*)
MONTHS	(*date-expr*)
MONTHS	(*date-expr*, *date-expr*)
MONTHS	(*date-expr*, *integer-expr*)
NOW	(*)
QUARTER	(*date-expr*)
SECOND	(*datetime-expr*)
SECONDS	(*datetime-expr*)
SECONDS	(*datetime-expr*, *datetime-expr*)
SECONDS	(*datetime-expr*, *integer-expr*)
TODAY	(*)
WEEKS	(*date-expr*)
WEEKS	(*date-expr*, *date-expr*)

Date and time functions	Parameters
WEEKS	(*date-expr*, *integer-expr*)
YEAR	(*date-expr*)
YEARS	(*date-expr*)
YEARS	(*date-expr*, *date-expr*)
YEARS	(*date-expr*, *integer-expr*)
YMD	(*year-num*, *month-num*, *day-num*)

Syntax 2

Table 4-5: Transact-SQL-compatible date and time functions

Transact-SQL compatible date and time functions	Parameters
DATEADD	(*date-part*, *numeric-expression*, *date-expr*)
DATEDIFF	(*date-part*, *date-expr1*, *date-expr2*)
DATENAME	(*date-part*, *date-expr*)
DATEPART	(*date-part*, *date-expr*)
GETDATE	()

Description

Sybase IQ provides two classes of date and time functions that can be used interchangeably, but have different styles. One set is Transact-SQL-compatible.

The date and time functions listed in Table 4-4 allow manipulation of time units. Most time units (such as MONTH) have four functions for time manipulation, although only two names are used (such as MONTH and MONTHS).

The functions listed in Table 4-5 are the Transact-SQL date and time functions. They allow an alternative way of accessing and manipulating date and time functions.

You should convert arguments to date functions to dates before used them. For example, this is incorrect:

```
days ( '1995-11-17', 2 )
```

This is correct:

```
days ( date( '1995-11-17' ), 2 )
```

Sybase IQ does not have the same constants or data type promotions as SQL Anywhere, with which it shares a common user interface. If you issue a SELECT statement without a FROM clause, the statement is passed to SQL Anywhere. The following statement is handled exclusively by SQL Anywhere:

```
SELECT WEEKS('1998/11/01');
```

The following statement, processed by Sybase IQ, uses a different starting point for the WEEKS function and returns a different result than the statement above:

```
SELECT WEEKS('1998/11/01') FROM iq_dummy;
```

Consider another example. The MONTHS function returns the number of months since an "arbitrary starting date." The "arbitrary starting date" of Sybase IQ, the imaginary date 0000-01-01, is chosen to produce the most efficient date calculations and is consistent across various data parts. SQL Anywhere does not have a single starting date. The following statements, the first processed by SQL Anywhere, the second by Sybase IQ, both return the answer 12:

```
SELECT MONTHS('0001/01/01');
SELECT MONTHS('0001/01/01') FROM iq_dummy;
```

However, also consider these statements:

```
SELECT DAYS('0001/01/01');
SELECT DAYS('0001/01/01') FROM iq_dummy;
```

The first, processed by SQL Anywhere, yields the value 307, but the second, processed by Sybase IQ, yields 166.

For the most consistent results, therefore, always include the table name in the FROM clause whether you need it or not.

Note Create a dummy table with only one column and row. You can then reference this table in the FROM clause for any SELECT statement that uses date or time functions, thus ensuring processing by Sybase IQ, and consistent results.

Date parts

Many of the date functions use dates built from date parts. Table 4-6 displays allowed values of *date-part*.

Table 4-6: Date part values

Date part	Abbreviation	Values
Year	yy	0001 – 9999
Quarter	qq	1 – 4
Month	mm	1 – 12
Week	wk	1 – 54
Day	dd	1 – 31
Dayofyear	dy	1 – 366
Weekday	dw	1 – 7 (Sun. – Sat.)
Hour	hh	0 – 23
Minute	mi	0 – 59
Second	ss	0 – 59
Millisecond	ms	0 – 999
Calyearofweek	cyr	Integer. The year in which the week begins. The week containing the first few days of the year can be part of the last week of the previous year, depending upon which day it begins. If the new year starts on a Thursday through Saturday, its first week starts on the last Sunday of the previous year. If the new year starts on a Sunday through Wednesday, none of its days are part of the previous year.
Calweekofyear	cwk	An integer from 1 to 54 representing the week number within the year that contains the specified date.
Caldayofweek	cdw	The day number within the week (Sunday = 1, Saturday = 7).

Note By default, Sunday is the first day of the week. To make Monday be the first day, use:

```
set option 'Date_First_Day_Of_Week' = '1'
```

For more information on specifying which day is the first day of the week, see DATE_FIRST_DAY_OF_WEEK option in *Reference: Statements and Options*.

Compatibility

For compatibility with Adaptive Server Enterprise, use the Transact-SQL date and time functions.

HTTP functions

Function

HTTP functions facilitate the handling of HTTP requests within Web services.

Table 4-7 lists all HTTP functions and their parameters.

Table 4-7: HTTP functions

HTTP function	Parameters
HTML_DECODE	(*string*)
HTML_ENCODE	(*string*)
HTTP_DECODE	(*string*)
HTTP_ENCODE	(*string*)
HTTP_VARIABLE	(*var-name* [[, *instance*], *header-field*])
NEXT_HTTP_HEADER	*header-name*
NEXT_HTTP_VARIABLE	*var-name*

Numeric functions

Function

Numeric functions perform mathematical operations on numerical data types or return numeric information.

Sybase IQ does not have the same constants or data type promotions as SQL Anywhere, with which it shares a common user interface. If you issue a SELECT statement without a FROM clause, the statement is passed through to SQL Anywhere. For the most consistent results, include the table name in the FROM clause whether you need it or not.

Note Consider creating a dummy table to use in such cases.

Table 4-8 lists numeric functions and their parameters.

Table 4-8: Numeric functions

Numeric function	Parameters
ABS	(*numeric-expr*)
ACOS	(*numeric-expr*)
ASIN	(*numeric-expr*)
ATAN	(*numeric-expr*)
ATAN2	(*numeric-expr1*, *numeric-expr2*)
CEIL	(*numeric-expr*)
CEILING	(*numeric-expr*)
COS	(*numeric-expr*)
COT	(*numeric-expr*)
DEGREES	(*numeric-expr*)
EXP	(*numeric-expr*)
FLOOR	(*numeric-expr*)
LN	(*numeric-expr*)
LOG	(*numeric-expr*)
LOG10	(*numeric-expr*)
MOD	(*dividend*, *divisor*)
PI	(*)
POWER	(*numeric-expr1*, *numeric-expr2*)
RADIANS	(*numeric-expr*)
RAND	([*integer-expr*])
REMAINDER	(*numeric-expr*, *numeric-expr*)
ROUND	(*numeric-expr*, *integer-expr*)
SIGN	(*numeric-expr*)
SIN	(*numeric-expr*)
SQRT	(*numeric-expr*)
SQUARE	(*numeric-expr*)
TAN	(*numeric-expr*)
“TRUNCATE”	(*numeric-expr*, *integer-expr*)
TRUNCNUM	(*numeric-expression*, *integer-expression*)
WIDTH_BUCKET	(*expression*, *min_value*, *max_value*, *num_buckets*)

String functions

Function

String functions perform conversion, extraction, or manipulation operations on strings, or return information about strings.

When working in a multibyte character set, check carefully whether the function being used returns information concerning characters or bytes.

Most of the string functions accept binary data (hexadecimal strings) in the *string-expr* parameter, but some of the functions, such as LCASE, UCASE, LOWER, and LTRIM, expect the string expression to be a character string.

Unless you supply a constant LENGTH argument to a function that produces a VARCHAR result (such as SPACE or REPEAT), the default length is the maximum allowed. See the “Field Size” column in Table 6-1 on page 348.

Sybase IQ queries containing one or more of these functions might return one of the following errors:

```
ASA Error -1009080: Key doesn't fit on a single database
page: 65560(4, 1)
```

```
ASA Error -1009119: Record size too large for database
page size
```

For example:

```
SELECT COUNT(*) FROM test1 a WHERE (a.col1 + SPACE(4-
LENGTH(a.col1)) + a.col2 + space(2- LENGTH(a.col2))) IN
(SELECT (b.col3) FROM test1 b);
```

To avoid such errors, cast the function result with an appropriate maximum length; for example:

```
SELECT COUNT(*) FROM test1 a WHERE (a.col1 +
CAST(SPACE(4-LENGTH(a.col1)) AS VARCHAR(4)) + a.col2 +
CAST(SPACE(2-LENGTH (a.col2)) AS VARCHAR(4))) IN
(SELECT (b.col3) FROM test1 b);
```

The errors are more likely with an IQ page size of 64K or a multibyte collation.

Table 4-9 lists string functions and their parameters.

Table 4-9: String functions

String function	Parameters
ASCII	(*string-expr*)
BIT_LENGTH	(*column-name*)
BYTE_LENGTH	(*string-expr*)
CHAR	(*integer-expr*)
CHAR_LENGTH	(*string-expr*)
CHARINDEX	(*string-expr1*, *string-expr2*)
DIFFERENCE	(*string-expr1*, *string-expr2*)
GRAPHICAL_PLAN	(*string-expr*)
HTML_PLAN	(*string-expr*)
INSERTSTR	(*numeric-expr*, *string-expr1*, *string-expr2*)
LCASE	(*string-expr*)
LEFT	(*string-expr*, *numeric-expr*)
LEN	(*string-expr*)
LENGTH	(*string-expr*)
LOCATE	(*string-expr1*, *string-expr2* [, *numeric-expr*])
LOWER	(*string-expr*)
LTRIM	(*string-expr*)
OCTET_LENGTH	(*column-name*)
PATINDEX	('%*pattern*%', *string_expr*)
REPEAT	(*string-expr*, *numeric-expr*)
REPLACE	(*original-string*, *search-string*, *replace-string*)
REVERSE	(*expression* | *uchar_expr*)
REPLICATE	(*string-expr*, *integer-expr*)
RIGHT	(*string-expr*, *numeric-expr*)
RTRIM	(*string-expr*)
SIMILAR	(*string-expr1*, *string-expr2*)
SORTKEY	(*string-expression* [, { *collation-id* | *collation-name* [(*collation-tailoring-string*)] }])
SOUNDEX	(*string-expr*)
SPACE	(*integer-expr*)
STR	(*numeric_expr* [, *length* [, *decimal*]])
STR_REPLACE	(*string_expr1*, *string_expr2*, *string_expr3*)
STRING	(*string1* [, *string2*, …, *string99*])
STUFF	(*string-expr1*, *start*, *length*, *string-expr2*)

String function	Parameters
SUBSTRING	(*string-expr*, *integer-expr* [, *integer-expr*])
TRIM	(*string-expr*)
UCASE	(*string-expr*)
UPPER	(*string-expr*)

System functions

Function

System functions return system information.

Table 4-10 lists the system functions and their parameters.

Table 4-10: System functions

System function	Parameters
COL_LENGTH	(*table-name*, *column-name*)
COL_NAME	(*table-id*, *column-id* [, *database-id*])
CONNECTION_PROPERTY	({ *property-id* | *property-name* } [, *connection-id*])
DATALENGTH	(*expression*)
DB_ID	([*database-name*])
DB_NAME	([*database-id*])
DB_PROPERTY	({ *property-id* | *property-name* } [,{*database-id* | *database-name* }])
EVENT_CONDITION	(*condition-name*)
EVENT_CONDITION_NAME	(*integer*)
EVENT_PARAMETER	(*context-name*)
GROUP_MEMBER	(*group-name-string-expression* [, *user-name-string-expression*])
INDEX_COL	(*table-name*, *index-id*, *key_#* [,*user-id*])
NEXT_CONNECTION	(*[connection-id] [, database-id]*)
NEXT_DATABASE	({ NULL | *database-id* })
OBJECT_ID	(*object-name*)
OBJECT_NAME	(*object-id* [, *database-id*])
PROPERTY	({ *property-id* | *property-name* })
PROPERTY_DESCRIPTION	(*property-id* | *property-name* })
PROPERTY_NAME	(*property-id*)
PROPERTY_NUMBER	(*property-name*)
SUSER_ID	([*user-name*])
SUSER_NAME	([*user-id*])
USER_ID	([*user-name*])
USER_NAME	([*user-id*])

Description

Databases currently running on a server are identified by a database name and a database ID number. The db_id and db_name functions provide information on these values.

A set of system functions provides information about properties of a currently running database, or of a connection, on the database server. These system functions take the database name or ID, or the connection name, as an optional argument to identify the database or connection for which the property is requested.

Performance

System functions are processed differently than other Sybase IQ functions. When queries to Sybase IQ tables include system functions, performance is reduced.

Compatibility

Table 4-11 shows the Adaptive Server Enterprise system functions and their status in Sybase IQ:

Table 4-11: Status of ASE system functions in Sybase IQ

Function	Status
col_length	Implemented
col_name	Implemented
db_id	Implemented
db_name	Implemented
index_col	Implemented
object_id	Implemented
object_name	Implemented
proc_role	Always returns 0
show_role	Always returns NULL
tsequal	Not implemented
user_id	Implemented
user_name	Implemented
suser_id	Implemented
suser_name	Implemented
datalength	Implemented
curunreservedpgs	Not implemented
data_pgs	Not implemented
host_id	Not implemented
host_name	Not implemented
lct_admin	Not implemented
reserved_pgs	Not implemented
rowcnt	Not implemented
used_pgs	Not implemented
valid_name	Not implemented
valid_user	Not implemented

Notes

- Some of the system functions are implemented in Sybase IQ as system stored procedures.
- The db_id, db_name, datalength, suser_id, and suser_name functions are implemented as built-in functions.

Connection properties

Connection properties apply to an individual connection. This section describes how to retrieve the value of a specific connection property or the values of all connection properties. For descriptions of all connection properties, see "Connection properties" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Configuring Your Database > Connection, database, and database server properties*.

Examples

❖ **Retrieving 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' )
```

❖ **Retrieving the values of all connection properties**

- Use the sa_conn_properties system procedure:

```
call sa_conn_properties
```

A separate row appears for each connection, for each property.

Properties available for the server

Server properties apply across the server as a whole. This section describes how to retrieve the value of a specific server property or the values of all server properties. For descriptions of all server properties, see "Database server properties" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Configuring Your Database > Connection, database, and database server properties*.

Examples

❖ **Retrieving the value of a server property**

- Use the property system function. The following statement returns the number of cache pages being used to hold the main heap:

```
select property ( 'MainHeapPages') from iq_dummy
```

❖ **Retrieving the values of all server properties**

- Use the sa_eng_properties system procedure:

```
call sa_eng_properties
```

Properties available for each database

Database properties apply to an entire database. This section describes how to retrieve the value of a specific database property or the values of all database properties. For descriptions of all of the database properties, see "Database server properties" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Configuring Your Database > Connection, database, and database server properties.*

Examples

❖ **Retrieving the value of a database property**

- Use the db_property system function. The following statement returns the page size of the current database:

```
select db_property ( 'PageSize') from iq_dummy
```

❖ **Retrieving the values of all database properties**

- Use the sa_db_properties system procedure:

```
call sa_db_properties
```

SQL and Java user-defined functions

There are two mechanisms for creating user-defined functions in Sybase IQ. You can use the SQL language to write the function, or you can use Java.

Note User-defined functions are processed by SQL Anywhere. They do not take advantage of the performance features of Sybase IQ. Queries that include user-defined functions run at least 10 times slower than queries without them.

In very few cases, differences in semantics between SQL Anywhere and Sybase IQ can produce different results for a query if it is issued in a user-defined function. For example, Sybase IQ treats the CHAR and VARCHAR data types as distinct and different, while SQL Anywhere treats CHAR data as if it were VARCHAR.

User-defined functions in SQL

You can implement your own functions in SQL using the CREATE FUNCTION statement. The RETURN statement inside the CREATE FUNCTION statement determines the data type of the function.

Once you have created a SQL user-defined function, you can use it anywhere a built-in function of the same data type is used.

Note Avoid using the CONTAINS predicate in a view that has a user-defined function, because the CONTAINS criteria is ignored. Use the LIKE predicate instead, or issue the query outside of a view.

For more information on creating SQL functions, see Chapter 1, "Using Procedures and Batches" in the *System Administration Guide: Volume 2*.

User-defined functions in Java

Although SQL functions are useful, Java classes provide a more powerful and flexible way of implementing user-defined functions, with the additional advantage that you can move them from the database server to a client application if desired.

Any **class method** of an installed Java class can be used as a user-defined function anywhere a built-in function of the same data type is used.

Instance methods are tied to particular instances of a class, and so have different behavior from standard user-defined functions.

For more information on creating Java classes, and on class methods, see "Java support in SQL Anywhere" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - Programming > Java in the database*.

Time series and forecasting functions

Function

Time series and forecasting functions are aggregate functions (see "Aggregate functions" on page 102) designed exclusively for financial time series analysis.

Note Time series and forecasting capability is available only with RAP – The Trading Edition™ Enterprise.

Table 4-12: Time series functions

Time series functions	Parameters
TS_ARMA_AR	(*timeseries_expression*, *ar_count*, *ar_elem*, *method*)
TS_ARMA_CONST	(*timeseries_expression*, *method*)
TS_ARMA_MA	(*timeseries_expression*, *ma_count*, *ma_elem*, *method*)
TS_AUTOCORRELATION	(*timeseries_expression*, *lagmax*, *lag_elem*)
TS_AUTO_UNI_AR	(*timeseries_expression*, *ar_count*, *ar_elem*, *method*)
TS_BOX_COX_XFORM	(*timeseries_expression*, *power* [, *shift* [, *inverse*]])
TS_DIFFERENCE	(*timeseries_expression*, *period1* [, *period2* [, *...period 10*]])
TS_ESTIMATE_MISSING	(*timeseries_expression*, *method*)
TS_LACK_OF_FIT	(*timeseries_expression*, *p_value*, *q_value*, *lagmax*, *[tolerance]*)
TS_LACK_OF_FIT_P	(*timeseries_expression*, *p_value*, *q_value*, *lagmax*, [*tolerance]*)
TS_MAX_ARMA_AR	(*timeseries_expression*, *ar_count*, *ar_elem*)
TS_MAX_ARMA_CONST	(*timeseries_expression*)
TS_MAX_ARMA_MA	(*timeseries_expression*, *ma_count*, *ma_elem*)
TS_MAX_ARMA_LIKELIHOOD	(*timeseries_expression*)
TS_OUTLIER_IDENTIFICATION	(*timeseries_expression*, *p_value*, *q_value*, *s_value*, *d_value*, [, *delta_value*[, *critical_value*]])
TS_PARTIAL_AUTOCORRELATION	(*timeseries_expression*, *lagmax*, *lag_elem*)
TS_VWAP	(*price_expression*, *volume_expression*)

Loading the IMSL libraries for time series and forecasting functions

The time series and forecasting functions call two third-party external libraries. The IMSL™ C Stat and C Math libraries, provided by Visual Numerics Inc., contain C functions used for financial time series and forecasting calculations.

The wrapper library **libtsudf** contains user-defined functions (UDFs) that invoke functions contained in the IMSL C Stat and C Math libraries.

The time series and forecasting SQL functions call the **libtsudf** wrapper library automatically. Sybase IQ loads the IMSL C Stat and C Math libraries when you call a valid user-defined aggregate function for time series and forecasting analysis.

The names and locations of the IMSL C Stat and C Math libraries vary depending on the platform on which Sybase IQ is installed:

Table 4-13: IMSL library locations and file names

	Windows 32-bit	Windows 64-bit	UNIX (except AIX)	AIX
Directory where libraries are located	bin32	bin64	lib64	lib64
Library file name	imslcmath_imsl_dll.dll imslcstat_imsl_dll.dll	imslcmath_imsl_dll.dll imslcstat_imsl_dll.dll	libimslcmath_imsl.so libimslcstat_imsl.so	libimslcmath_imsl_r.so libimslcstat_imsl_r.so

IMSL library time series function error-handling

You can control the error-handling behavior for the time series functions that call the IMSL libraries. If a runtime error occurs when invoking IMSL library functions, Sybase IQ responds according to your error-handling choice. You can choose from four error-handling options, ranging from ignoring all errors and warnings (the default behavior), to specifying which level of error severity causes Sybase IQ to abort a SQL statement and return an error message.

Use the following SQL statement to control error-handling:

```
set option PUBLIC.Time_Series_Error_Level = '<value>'
```

Valid values are:

Table 4-14: IMSL library time series function error-handling

Value	Description
0 (default)	All types of warnings and errors that can be obtained while invoking an IMSL library function are ignored. When such a condition is encountered, the time series function returns a NULL value.
1	If the time series function obtains a warning or an error message while invoking an IMSL library function, IQ returns an error message and aborts the SQL query.
2	If the time series function obtains a fatal error message while invoking an IMSL library function, IQ returns an error message and aborts the SQL query. However if a warning is obtained then the time series function returns a NULL value.
3	If the time series function obtains a terminal error message while invoking an IMSL library function, IQ returns an error message and aborts the SQL query. However if a warning or a fatal error is obtained then the time series function returns a NULL value.

IMSL library time series function error logging

You can control how Sybase IQ logs the error messages returned by IMSL library time series function error-handling. Four options exist for logging error messages to the log file.

Use the following SQL statement to control error logging:

```
set option PUBLIC.Time_Series_Log_Level = ‘<value>’
```

Table 4-15: IMSL library time series function error logging

Value	Description
0 (default)	All warnings and errors returned while invoking an IMSL library function are ignored and not logged to the log file.
1	If the time series function returns a warning or an error message while invoking an IMSL library function, a message is logged to the log file.
2	If the time series function returns a fatal error message while invoking an IMSL library function, a message is logged to the log file. Warnings are not logged.
3	If the time series function returns a terminal error message while invoking an IMSL library function, a message is logged to the log file. Warnings and fatal errors are not logged.

Miscellaneous functions

Function

Miscellaneous functions perform operations on arithmetic, string, or date/time expressions, including the return values of other functions.

Table 4-16 lists the miscellaneous functions and their parameters.

Table 4-16: Miscellaneous functions

Miscellaneous functions	Parameters
ARGN	(*integer-expr*, *expression* [, …)
COALESCE	(*expression*, *expression* [, *expression* …])
IFNULL	(*expression1*, *expression2* [, *expression3*])
ISNULL	(*expression*, *expression* [, *expression* …])
NULLIF	(*expression1*, *expression2*)
NUMBER	(*)
ROWID	(*table-name*)

Compatibility

Adaptive Server Enterprise supports only the COALESCE, ISNULL, and NULLIF functions.

Alphabetical list of functions

This section describes each function individually. The function type, for example, Numeric or String, is indicated in brackets next to the function name.

Some of the results in the examples have been rounded or truncated.

The actual values of database object IDs, such as the object ID of a table or the column ID of a column, might differ from the values shown in the examples.

ABS function [Numeric]

Function
Returns the absolute value of a numeric expression.

Syntax
ABS (*numeric-expression*)

Parameters
numeric-expression The number whose absolute value is to be returned.

Example
The following statement returns the value 66:

```
SELECT ABS( -66 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

ACOS function [Numeric]

Function
Returns the arc-cosine, in radians, of a numeric expression.

Syntax
ACOS (*numeric-expression*)

Parameters
numeric-expression The cosine of the angle.

Example
The following statement returns the value 1.023945:

```
SELECT ACOS( 0.52 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also
"ASIN function [Numeric]" on page 129

"ATAN function [Numeric]" on page 130

"ATAN2 function [Numeric]" on page 130

"COS function [Numeric]" on page 143

ARGN function [Miscellaneous]

Function

Returns a selected argument from a list of arguments.

Syntax

ARGN (*integer-expression*, *expression* [, …])

Parameters

integer-expression The position of an argument within a list of expressions.

expression An expression of any data type passed into the function. All supplied expressions must be of the same data type.

Example

The following statement returns the value 6:

```
SELECT ARGN( 6, 1,2,3,4,5,6 ) FROM iq_dummy
```

Usage

Using the value of *integer-expression* as *n* returns the *n*th argument (starting at 1) from the remaining list of arguments. While the expressions can be of any data type, they must all be of the same data type. The integer expression must be from one to the number of expressions in the list or NULL is returned. Multiple expressions are separated by a comma.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

ASCII function [String]

Function

Returns the integer ASCII value of the first byte in a string-expression.

Syntax

ASCII (*string-expression*)

Parameters

string-expression The string

Example

The following statement returns the value 90, when the collation sequence is set to the default ISO_BINENG:

```
SELECT ASCII( 'Z' ) FROM iq_dummy
```

Usage

If the string is empty, ASCII returns zero. Literal strings must be enclosed in quotes.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

ASIN function [Numeric]

Function

Returns the arc-sine, in radians, of a number.

Syntax **ASIN** (*numeric-expression*)

Parameters **numeric-expression** The sine of the angle

Example The following statement returns the value 0.546850.

```
SELECT ASIN( 0.52 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also "ACOS function [Numeric]" on page 128

"ATAN function [Numeric]" on page 130

"ATAN2 function [Numeric]" on page 130

"SIN function [Numeric]" on page 250

ATAN function [Numeric]

Function Returns the arc-tangent, in radians, of a number.

Syntax **ATAN** (*numeric-expression*)

Parameters **numeric-expression** The tangent of the angle

Example The following statement returns the value 0.479519:

```
SELECT ATAN( 0.52 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also "ACOS function [Numeric]" on page 128

"ASIN function [Numeric]" on page 129

"ATAN2 function [Numeric]" on page 130

"TAN function [Numeric]" on page 267

ATAN2 function [Numeric]

Function Returns the arc-tangent, in radians, of the ratio of two numbers.

Syntax **ATAN2** (*numeric-expression1*, *numeric-expression2*)

Parameters

numeric-expression1 The numerator in the ratio whose arc tangent is calculated.

numeric-expression2 The denominator in the ratio whose arc-tangent is calculated.

Example

The following statement returns the value 0.00866644968879073143:

```
SELECT ATAN2( 0.52, 060 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** ATAN2 is not supported by Adaptive Server Enterprise

See also

“ACOS function [Numeric]” on page 128

“ASIN function [Numeric]” on page 129

“ATAN function [Numeric]” on page 130

“TAN function [Numeric]” on page 267

AVG function [Aggregate]

Function

Computes the average of a numeric expression for a set of rows, or computes the average of a set of unique values.

Syntax

AVG (*numeric-expression* | **DISTINCT** *column-name*)

Parameters

numeric-expression The value whose average is calculated over a set of rows.

DISTINCT column-name Computes the average of the unique values in *column-name*. This is of limited usefulness, but is included for completeness.

Example

The following statement returns the value 49988.6:

```
SELECT AVG ( salary ) FROM Employees
```

Usage

This average does not include rows where *numeric -expression* is the NULL value. Returns the NULL value for a group containing no rows.

Standards and compatibility

- **SQL92** SQL92 compatible.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

“COUNT function [Aggregate]” on page 146

“SUM function [Aggregate]” on page 265

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

BIGINTTOHEX function [Data type conversion]

Function
Returns the hexadecimal equivalent in VARCHAR(16) of a decimal integer.

Syntax
BIGINTTOHEX (*integer-expression*)

Parameters
integer-expression The integer to be converted to hexadecimal.

Examples
The following statement returns the value `0000000000000009`:

```
SELECT BIGINTTOHEX(9) FROM iq_dummy
```

The following statement returns the value `FFFFFFFFFFFFFFF7`:

```
SELECT BIGINTTOHEX (-9) FROM iq_dummy
```

Usage
The BIGINTTOHEX function accepts an integer expression that evaluates to BIGINT and returns the hexadecimal equivalent. Returned values are left appended with zeros up to a maximum of 16 digits. All types of unscaled integer data types are accepted as integer expressions.

Conversion is done automatically, if required. Constants are truncated, only if the fraction values are zero. A column cannot be truncated, if the column is declared with a positive scale value. If conversion fails, Sybase IQ returns an error unless the CONVERSION_ERROR option is OFF. In that case, the result is NULL.

Standards and compatibility
- **SQL92** Transact-SQL extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also
CONVERSION_ERROR option [TSQL] in *Reference: Statements and Options*

"HEXTOBIGINT function [Data type conversion]" on page 178

"HEXTOINT function [Data type conversion]" on page 179

"INTTOHEX function [Data type conversion]" on page 189

BIT_LENGTH function [String]

Function
Returns an unsigned 64-bit value containing the bit length of the column parameter.

Syntax
BIT_LENGTH(*column-name*)

Parameters
column-name The name of a column

Usage
The return value of a NULL argument is NULL.

The BIT_LENGTH function supports all Sybase IQ data types.

Standards and compatibility

Sybase Not supported by SQL Anywhere or Adaptive Server Enterprise

See also

"OCTET_LENGTH function [String]" on page 216

BYTE_LENGTH function [String]

Function

Returns the number of bytes in a string.

Syntax

BYTE_LENGTH (*string-expression*)

Parameters

string-expression The string whose length is to be calculated

Example

The following statement returns the value 12:

```
SELECT BYTE_LENGTH( 'Test Message' ) FROM iq_dummy
```

Usage

Trailing white space characters are included in the length returned.

The return value of a NULL string is NULL.

If the string is in a multibyte character set, the BYTE_LENGTH value differs from the number of characters returned by CHAR_LENGTH.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"CHAR_LENGTH function [String]" on page 136

"DATALENGTH function [System]" on page 148

"LENGTH function [String]" on page 197

CAST function [Data type conversion]

Function

Returns the value of an expression converted to a supplied data type.

Syntax

CAST (*expression* **AS** *data type*)

Parameters

expression The expression to be converted

data type The target data type

Examples

The following function ensures a string is used as a date:

```
CAST( '2000-10-31' AS DATE )
```

The value of the expression 1 + 2 is calculated, and the result cast into a single-character string, the length the data server assigns:

```
CAST( 1 + 2 AS CHAR )
```

You can use the CAST function to shorten strings:

```
SELECT CAST( lname AS CHAR(5) ) FROM Customers
```

Usage

If you do not indicate a length for character string types, Sybase IQ chooses an appropriate length. If neither precision nor scale is specified for a DECIMAL conversion, the database server selects appropriate values.

If neither precision nor scale is specified for the explicit conversion of NULL to NUMERIC, the default is NUMERIC(1,0). For example,

```
SELECT CAST( NULL AS NUMERIC ) A,
       CAST( NULL AS NUMERIC(15,2) ) B
```

is described as:

```
A NUMERIC(1,0)
B NUMERIC(15,2)
```

Standards and compatibility

- **SQL92** This function is SQL92 compatible.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

"CONVERT function [Data type conversion]" on page 139

CEIL function [Numeric]

Function

Returns the smallest integer greater than or equal to the specified expression.

CEIL is as synonym for CEILING.

Syntax

CEIL (*numeric-expression*)

Parameters

expression A column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, CEIL generates an error. The return value has the same data type as the value supplied.

Usage

For a given expression, the CEIL function takes one argument. For example, CEIL (-123.45) returns -123. CEIL (123.45) returns 124.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

"CEILING function [Numeric]" on page 135

Chapter 11, "International Languages and Character Sets" in the *System Administration Guide: Volume 1*

CEILING function [Numeric]

Function
Returns the ceiling (smallest integer not less than) of a number.

CEIL is as synonym for CEILING.

Syntax
CEILING (*numeric-expression*)

Parameters
numeric-expression The number whose ceiling is to be calculated

Examples
The following statement returns the value 60.00000:

```
SELECT CEILING( 59.84567 ) FROM iq_dummy
```

The following statement returns the value 123:

```
SELECT CEILING( 123 ) FROM iq_dummy
```

The following statement returns the value 124.00:

```
SELECT CEILING( 123.45 ) FROM iq_dummy
```

The following statement returns the value -123.00:

```
SELECT CEILING( -123.45 ) FROM iq_dummy
```

Standards and compatibility
- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also
"FLOOR function [Numeric]" on page 173

CHAR function [String]

Function
Returns the character with the ASCII value of a number.

Syntax
CHAR (*integer-expression*)

Parameters
integer-expression The number to be converted to an ASCII character. The number must be in the range 0 to 255, inclusive.

Examples
The following statement returns the value "Y":

```
SELECT CHAR( 89 ) FROM iq_dummy
```

The following statement returns the value "S":

```
SELECT CHAR( 83 ) FROM iq_dummy
```

Usage

The character in the current database character set corresponding to the supplied numeric expression modulo 256 is returned.

CHAR returns NULL for integer expressions with values greater than 255 or less than zero.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

CHAR_LENGTH function [String]

Function

Returns the number of characters in a string.

Syntax

CHAR_LENGTH (*string-expression*)

Parameters

string-expression The string whose length is to be calculated

Usage

Trailing white space characters are included in the length returned.

The return value of a NULL string is NULL.

If the string is in a multibyte character set, the CHAR_LENGTH value may be less than the BYTE_LENGTH value.

Example

The following statement returns the value 8:

```
SELECT CHAR_LENGTH( 'Chemical' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** This function is SQL92 compatible
- **Sybase** Compatible with Adaptive Server Enterprise

See also

“BYTE_LENGTH function [String]” on page 133

CHARINDEX function [String]

Function

Returns the position of the first occurrence of a specified string in another string.

Syntax

CHARINDEX (*string-expression1, string-expression2*)

Parameters

string-expression1 The string for which you are searching. This string is limited to 255 bytes.

string-expression2 The string to be searched. The position of the first character in the string being searched is 1.

Example

The statement:

```
SELECT Surname, GivenName
FROM Employees
WHERE CHARINDEX('K', Surname ) = 1
```

returns the following values:

Surname	**GivenName**
Klobucher	James
Kuo	Felicia
Kelly	Moira

Usage

All the positions or offsets, returned or specified, in the CHARINDEX function are always character offsets and may be different from the byte offset for multibyte data.

If the string being searched contains more than one instance of the specified string, CHARINDEX returns the position of the first instance.

If the string being searched does not contain the specified string, CHARINDEX returns zero (0).

Searching for a zero-length string returns 1.

If any of the arguments is NULL, the result is NULL.

CHARINDEX returns a 32 bit signed integer position for CHAR and VARCHAR columns.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

"SUBSTRING function [String]" on page 265

Chapter 4, "Function Support" in *Large Objects Management in Sybase IQ*

COALESCE function [Miscellaneous]

Function

Returns the first non-NULL expression from a list.

Syntax

COALESCE (*expression*, *expression* [, …])

Parameters

expression Any expression

Example

The following statement returns the value 34:

```
SELECT COALESCE( NULL, 34, 13, 0 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** SQL92

- **Sybase** Compatible with Adaptive Server Enterprise.

COL_LENGTH function [System]

Function

Returns the defined length of a column.

Syntax

COL_LENGTH (*table-name*, *column-name*)

Parameters

table-name The table name

column-name The column name

Example

The following statement returns the column length 35:

```
SELECT COL_LENGTH ( 'CUSTOMERS', 'ADDRESS' ) FROM
iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ

See also

"DATALENGTH function [System]" on page 148

COL_NAME function [System]

Function

Returns the column name.

Syntax

COL_NAME (*table-id*, *column-id* [, *database-id*])

Parameters

table-id The object ID of the table

column-id The column ID of the column

database-id The database ID

Examples

The following statement returns the column name lname. The object ID of the Customers table is 100209, as returned by the OBJECT_ID function. The column ID is stored in the column_id column of the syscolumn system table. The database ID of the iqdemo database is 0, as returned by the DB_ID function.

```
SELECT COL_NAME( 100209, 3, 0 ) FROM iq_dummy
```

The following statement returns the column name city.

```
SELECT COL_NAME ( 100209, 5 )FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ

See also

"DB_ID function [System]" on page 161

"OBJECT_ID function [System]" on page 215

CONNECTION_PROPERTY function [System]

Function
Returns the value of a given connection property as a string.

Syntax

CONNECTION_PROPERTY ({ *integer-expression1* | *string-expression* } … [, *integer-expression2*])

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

integer-expression1 In most cases, it is more convenient to supply a string expression as the first argument. If you do supply *integer-expression1*, it is the connection property ID. You can determine this using the PROPERTY_NUMBER function.

string-expression The connection property name. You must specify either the property ID or the property name.

integer-expression2 The connection ID of the current database connection. The current connection is used if this argument is omitted.

Example
The following statement returns the number of prepared statements being maintained, for example, 4:

```
SELECT connection_property( 'PrepStmt' )FROM iq_dummy
```

Usage
The current connection is used if the second argument is omitted.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

"Connection properties" on page 121

"PROPERTY_NUMBER function [System]" on page 227

CONVERT function [Data type conversion]

Function
Returns an expression converted to a supplied data type.

Syntax

CONVERT (*data-type*, *expression* [, *format-style*])

Parameters

data-type The data type to which the expression is converted

expression The expression to be converted

format-style For converting strings to date or time data types and vice versa, *format-style* is a style code number that describes the date format string to be used. Table 4-17 lists the meanings of the values of the *format-style* argument.

Table 4-17: CONVERT format style code output

Without century (yy)	With century (yyyy)	Output
-	0 or 100	mmm dd yyyy hh:nnAM (or PM)
1	101	mm/dd/yy[yy]
2	102	[yy]yy.mm.dd
3	103	dd/mm/yy[yy]
4	104	dd.mm.yy[yy]
5	105	dd-mm-yy[yy]
6	106	dd mmm yy[yy]
7	107	mmm dd, yy[yy]
8	108	hh:nn:ss
-	9 or 109	mmm dd yyyy hh:nn:ss:sssAM (or PM)
10	110	mm-dd-yy[yy]
11	111	[yy]yy/mm/dd
12	112	[yy]yymmdd
13	113	dd mmm yyyy hh:nn:ss:sss (24 hour clock, Europe default + milliseconds, 4-digit year)
14	114	hh:nn:ss (24 hour clock)
20	120	yyyy-mm-dd hh:nn:ss (24-hour clock, ODBC canonical, 4-digit year)
21	121	yyyy-mm-dd hh:nn:ss.sss (24 hour clock, ODBC canonical with milliseconds, 4-digit year)
-	365	yyyyjjj (as a string or integer, where jjj is the Julian day number from 1 to 366 within the year)

If no *format-style* argument is provided, style code 0 is used.

Examples

The following statements illustrate the use of format styles:

```
SELECT CONVERT( CHAR( 20 ), order_date, 104 )
FROM sales_order
```

order_date
16.03.1993

order_date
20.03.1993
23.03.1993
25.03.1993
...

```
SELECT CONVERT( CHAR( 20 ), order_date, 7 )
FROM sales_order
```

order_date
mar 16, 93
mar 20, 93
mar 23, 93
mar 25, 93
...

The following statements illustrate the use of the format style 365, which converts data of type DATE and DATETIME to and from either string or integer type data:

```
CREATE TABLE tab
   (date_col DATE, int_col INT, char7_col CHAR(7));
INSERT INTO tab (date_col, int_col, char7_col)
   VALUES (‘Dec 17, 2004’, 2004352, ‘2004352’);
```

`SELECT CONVERT(VARCHAR(8), tab.date_col, 365) FROM tab;`
returns `‘2004352’`

`SELECT CONVERT(INT, tab.date_col, 365) from tab;`
returns `2004352`

`SELECT CONVERT(DATE, tab.int_col, 365) FROM TAB;`
returns `2004-12-17`

`SELECT CONVERT(DATE, tab.char7_col, 365) FROM tab;`
returns `2004-12-17`

The following statement illustrates conversion to an integer, and returns the value 5.

```
SELECT CONVERT( integer, 5.2 ) FROM iq_dummy
```

Usage

The result data type of a CONVERT function is a LONG VARCHAR. If you use CONVERT in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set CONVERT to the correct data type and size.

See “REPLACE function [String]” for more information.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise and SQL Anywhere, except for format style 365, which is an IQ-only extension

See also

“CAST function [Data type conversion]” on page 133

CORR function [Aggregate]

Function

Returns the correlation coefficient of a set of number pairs.

Syntax 1

CORR (*dependent-expression*, *independent-expression*)

Syntax 2

CORR (*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

dependent-expression The variable that is affected by the independent-expression.

independent-expression The variable that influences the outcome.

Example

The following example performs a correlation to discover whether age is associated with income level. This function returns the value 0.440227:

```
SELECT CORR( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate
) ) ) FROM Employees;
```

Usage

The CORR function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then CORR returns NULL.

dependent-expression and *independent-expression* are both numeric. The function is applied to the set of (*dependent-expression*, *independent-expression*) after eliminating the pairs for which either *dependent-expression* or *independent-expression* is NULL. The following computation is made:

COVAR_POP (y, x) / (STDDEV_POP (x) * STDDEV_POP (y))

where x represents the *dependent-expression* and y represents the *independent-expression*.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Standards and compatibility

- **SQL2008** SQL foundation feature outside of core SQL
- **Sybase** Compatible with SQL Anywhere

COS function [Numeric]

Function

Returns the cosine of a number, expressed in radians.

Syntax

COS (*numeric-expression*)

Parameters

numeric-expression The angle, in radians.

Example

The following statement returns the value 0.86781:

```
SELECT COS( 0.52 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

“ACOS function [Numeric]” on page 128

“COT function [Numeric]” on page 143

“SIN function [Numeric]” on page 250

“TAN function [Numeric]” on page 267

COT function [Numeric]

Function

Returns the cotangent of a number, expressed in radians.

Syntax

COT (*numeric-expression*)

Parameters

numeric-expression The angle, in radians.

Example

The following statement returns the value 1.74653:

```
SELECT COT( 0.52 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

“COS function [Numeric]” on page 143

"SIN function [Numeric]" on page 250

"TAN function [Numeric]" on page 267

COVAR_POP function [Aggregate]

Function

Returns the population covariance of a set of number pairs.

Syntax 1

COVAR_POP (*dependent-expression*, *independent-expression*)

Syntax 2

COVAR_POP (*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Example

The following example measures the strength of association between employee age and salary. This function returns the value 73785.840059:

```
SELECT COVAR_POP( Salary, ( YEAR( NOW( ) ) - YEAR(
BirthDate ) ) ) FROM Employees;
```

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then COVAR_POP returns NULL.

dependent-expression and *independent-expression* are both numeric. The function is applied to the set of (*dependent-expression*, *independent-expression*) after eliminating the pairs for which either *dependent-expression* or *independent-expression* is NULL. The following computation is made:

(SUM(x*y) - SUM(x) * SUM(y) / n) / n

where x represents the *dependent-expression* and y represents the *independent-expression*.

See "Mathematical formulas for the aggregate functions" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions*.

.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see "Analytical functions" on page 104.

Standards and compatibility

- **SQL2008** SQL foundation feature outside of core SQL
- **Sybase** Compatible with SQL Anywhere

COVAR_SAMP function [Aggregate]

Function

Returns the sample covariance of a set of number pairs.

Syntax 1

COVAR_SAMP (*dependent-expression*, *independent-expression*)

Syntax 2

COVAR_SAMP (*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Example

The following example measures the strength of association between employee age and salary. This function returns the value 74782.946005:

```
SELECT COVAR_SAMP( Salary, ( 2008 - YEAR( BirthDate ) )
) FROM Employees;
```

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then COVAR_SAMP returns NULL.

Both *dependent-expression* and *independent-expression* are numeric. The function is applied to the set of (*dependent-expression*, *independent-expression*) after eliminating the pairs for which either *dependent-expression* or *independent-expression* is NULL.

(SUM(x*y) - SUM(x) * SUM(y) / n) / (n-1)

where x represents the *dependent-expression* and y represents the *independent-expression*.

See "Mathematical formulas for the aggregate functions" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions*.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see "Analytical functions" on page 104.

Standards and compatibility

- **SQL2008** SQL foundation feature outside of core SQL
- **Sybase** Compatible with SQL Anywhere

COUNT function [Aggregate]

Function

Counts the number of rows in a group, depending on the specified parameters.

Syntax

COUNT (* | *expression* | **DISTINCT** *column-name*)

Parameters

***** Returns the number of rows in each group.

expression Returns the number of rows in each group where *expression* is not the NULL value.

DISTINCT column-name Returns the number of different values in *column-name*. Rows where the value is the NULL value are not included in the count.

Example

The following statement returns each unique city, and the number of rows with that city value:

```
SELECT city , Count(*)
FROM Employees
GROUP BY city
```

Standards and compatibility

- **SQL92** SQL92 compatible.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

"AVG function [Aggregate]" on page 131

"SUM function [Aggregate]" on page 265

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

CUME_DIST function [Ranking]

Function

The CUME_DIST function calculates the relative position of one value among a group of rows. It returns a decimal value between 0 and 1.

Syntax

CUME_DIST () OVER (*window-spec*)

window-spec: See the Usage section below.

Example

The following example returns a result set that provides a cumulative distribution of the salaries of employees who live in California:

```
SELECT DepartmentID, Surname, Salary,
CUME_DIST() OVER (PARTITION BY DepartmentID
ORDER BY Salary DESC) "Rank"
FROM Employees
WHERE State IN ('CA');
```

The returned result set is:

Table 4-18: CUME_DIST result set

DepartmentID	Surname	Salary	Rank
200	Savarino	72300.000	0.333333
200	Clark	45000.000	0.666667
200	Overbey	39300.000	1.000000

Usage

Sybase IQ calculates the cumulative distribution of a value of x in a set S of size N using:

CUME_DIST(x) = *number of values in S coming before and including x in the specified order / N*

Composite sort-keys are not currently allowed in the CUME_DIST function. You can use composite sort-keys with any of the other rank functions.

You can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. The *window-spec* must contain the ORDER BY clause, and cannot contain a ROWS or RANGE clause. For information on how to specify the window, see “Analytical functions” on page 104.

Note DISTINCT is not supported.

Standards and compatibility

- **SQL2008** SQL/OLAP feature T612
- **Sybase** Compatible with SQL Anywhere

DATALENGTH function [System]

Function

Returns the length of the expression in bytes.

Syntax

DATALENGTH (*expression*)

Parameters

expression The expression is usually a column name. If the expression is a string constant, it must be enclosed in quotes.

Usage

Table 4-19 lists the return values of DATALENGTH.

Table 4-19: DATALENGTH return values

Data type	DATALENGTH
SMALLINT	2
INTEGER	4
DOUBLE	8
CHAR	Length of the data
BINARY	Length of the data

Example

The following statement returns the value 35, the longest string in the company_name column:

```
SELECT MAX( DATALENGTH( company_name ) )
FROM Customers
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ

See also

“CHAR_LENGTH function [String]” on page 136

“COL_LENGTH function [System]” on page 138

DATE function [Date and time]

Function

Converts the expression into a date, and removes any hours, minutes, or seconds.

Syntax

DATE (*expression*)

Parameters

expression The value to be converted to date format. The expression is usually a string.

Example

The following statement returns the value 1988-11-26 as a date.

```
SELECT DATE( '1988-11-26 21:20:53' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

DATEADD function [Date and time]

Function

Returns the date produced by adding the specified number of the specified date parts to a date.

Syntax

DATEADD (*date-part*, *numeric-expression*, *date-expression*)

Parameters

date part The date part to be added to the date.

For a complete listing of allowed date parts, see “Date parts” on page 112.

numeric-expression The number of *date parts* to be added to the date. The *numeric-expression* can be any numeric type; the value is truncated to an integer.

date-expression The date to be modified.

Example

The following statement returns the value 1995-11-02 00:00:00.000:

```
SELECT DATEADD( month, 102, '1987/05/02' ) FROM iq_dummy
```

Usage

DATEADD is a Transact-SQL compatible data manipulation function.

Standards and compatibility

- **SQL92** Transact-SQL extension.
- **Sybase** Compatible with Adaptive Server Enterprise.

DATECEILING function [Date and time]

Function

Calculates a new date, time, or datetime value by increasing the provided value up to the nearest larger value of the specified multiple with the specified granularity.

Syntax

DATECEILING (*date-part*, *datetime-expression [*,*multiple -expression]*)

Parameters

date part The date part to be added to the date.

The following date parts are not compatible with DATECEILING:

- DayofYear
- WeekDay
- CalYearofWeek
- CalWeekofYear
- CalDayofWeek

For a complete listing of date parts, see Date part values on page 113.

datetime-expression The date, time, or date-time expression containing the value you are evaluating.

multiple-expression (Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by the date_part parameter to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 10-minute intervals. Note that if multiple-expression evaluates to zero, evaluates to a negative number, or is an explicit NULL constant, Sybase IQ generates an error. If this multiple-expression evaluates to a NULL, then the function result is NULL.

Example

This returns the value August 13, 2009 10:40.00.000AM:

```
SELECT DATECEILING( MI, 'August 13, 2009,
10:32.00.132AM', 10) FROM iq_dummy
```

Usage

This function calculates a new date, time, or datetime value by increasing the provided value up to the nearest larger value with the specified granularity. If you include the optional multiple-expression parameter, then the function increases the date and time up to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the multiple-expression parameter.

If you specify a multiple-expression for the millisecond, second, minute, or hour date parts, IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day

For example, if you specify a multiple of two minutes, IQ applies two-minute intervals starting at the current hour.

For the millisecond, second, minute, and hour date parts, specify a multiple-expression value that divides evenly into the range of the specified date part:

- For milliseconds, the valid multiple-expression values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For seconds and minutes, the valid multiple-expression values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60
- For hours, the valid multiple-expression values are: 1, 2, 3, 4, 6, 8, 12, 24

If you specify a multiple-expression for the day, week, month, quarter, or year date parts, IQ assumes the intervals started at the smallest date value (0000-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0000-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, then Sybase IQ calculates 10-day intervals starting at 0000-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a multiple that divides evenly into the next larger unit of time granularity.

If IQ rounds to a multiple of the week date part, the date value is always Sunday.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere

DATEDIFF function [Date and time]

Function

Returns the interval between two dates.

Syntax

DATEDIFF (*date-part*, *date-expression1*, *date-expression2*)

Parameters

date-part Specifies the date part in which the interval is to be measured.

For a complete listing of allowed date parts, see “Date parts” on page 112.

date-expression1 The starting date for the interval. This value is subtracted from *date-expression2* to return the number of date parts between the two arguments.

date-expression2 The ending date for the interval. *date-expression1* is subtracted from this value to return the number of date parts between the two arguments.

Examples

The following statement returns 1:

```
SELECT DATEDIFF( hour, '4:00AM', '5:50AM' ) FROM
iq_dummy
```

The following statement returns 102:

```
SELECT DATEDIFF( month, '1987/05/02', '1995/11/15' )
FROM iq_dummy
```

The following statement returns 0:

```
SELECT DATEDIFF( day, '00:00', '23:59' ) FROM iq_dummy
```

The following statement returns 4:

```
SELECT DATEDIFF( day, '1999/07/19 00:00', '1999/07/23
23:59' ) FROM iq_dummy
```

The following statement returns 0:

```
SELECT DATEDIFF( month, '1999/07/19', '1999/07/23' )
FROM iq_dummy
```

The following statement returns 1:

```
SELECT DATEDIFF( month, '1999/07/19', '1999/08/23' )
FROM iq_dummy
```

Usage

This function calculates the number of date parts between two specified dates. The result is a signed integer value equal to (date2 - date1), in date parts.

DATEDIFF results are truncated, not rounded, when the result is not an even multiple of the date part.

When you use day as the date part, DATEDIFF returns the number of midnights between the two times specified, including the second date, but not the first. For example, the following statement returns the value 5. Midnight of the first day 2003/08/03 is not included in the result. Midnight of the second day *is* included, even though the time specified is before midnight.

```
SELECT DATEDIFF( day, '2003/08/03 14:00', '2003/08/08
14:00' ) FROM iq_dummy
```

When you use month as the date part, DATEDIFF returns the number of first-of-the-months between two dates, including the second date but not the first. For example, both of the following statements return the value 9:

```
SELECT DATEDIFF( month, '2003/02/01', '2003/11/15' )
FROM iq_dummy;
SELECT DATEDIFF( month, '2003/02/01', '2003/11/01' )
FROM iq_dummy;
```

The first date 2003/02/01 is a first-of-month, but is not included in the result of either query. The second date 2003/11/01 in the second query is also a first-of-month and *is* included in the result.

When you use week as the date part, DATEDIFF returns the number of Sundays between the two dates, including the second date but not the first. For example, in the month 2003/08, the dates of the Sundays are 03, 10, 17, 24, and 31. The following query returns the value 4:

```
SELECT DATEDIFF( week, '2003/08/03', '2003/08/31' )
FROM iq_dummy;
```

The first Sunday (2003/08/03) is not included in the result.

For smaller time units, there are overflow values:

- **milliseconds** 24 days.
- **seconds** 68 years.
- **minutes** 4083 years.
- **others** No overflow limit

.

The function returns an overflow error if you exceed these limits.

Standards and compatibility

- **SQL92** Transact-SQL extension.
- **Sybase** Compatible with Adaptive Server Enterprise.

DATEFLOOR function [Date and time]

Function

Calculates a new date, time, or datetime value by reducing the provided value down to the nearest lower value of the specified multiple with the specified granularity.

Syntax

DATEFLOOR (*date-part*, *datetime-expression [,multiple-expression]*)

Parameters

date part The date part to be added to the date.

The following date parts are not compatible with DATEFLOOR:

- DayofYear
- WeekDay
- CalYearofWeek
- CalWeekofYear
- CalDayofWeek

For a complete listing of date parts, see Date part values on page 113.

datetime-expression The date, time, or date-time expression containing the value you are evaluating.

multiple-expression (Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by the date_part parameter to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 10-minute intervals. Note that if multiple-expression evaluates to zero, evaluates to a negative number, or is an explicit NULL constant, then IQ generates an error. If multiple-expression evaluates to a NULL, then the function result is NULL.

Example

The following statement returns the value August 13, 2009 10:35:00.000AM:

```
SELECT DATEFLOOR( MINUTE, 'August 13, 2009
10:35.22.123AM') FROM iq_dummy
```

Usage

This function calculates a new date, time, or datetime value by reducing the provided value down to the nearest lower value with the specified granularity. If you include the optional multiple-expression parameter, then the function reduces the date and time down to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the multiple-expression parameter.

If you specify a multiple-expression for the millisecond, second, minute, or hour date parts, IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day

For example, if you specify a multiple of two minutes, IQ applies two minute intervals starting at the current hour.

For the millisecond, second, minute, and hour date parts, specify a multiple-expression value that divides evenly into the range of the specified date part:

- For milliseconds, the valid multiple-expression values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For seconds and minutes, the valid multiple-expression values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60
- For hours, the valid multiple-expression values are: 1, 2, 3, 4, 6, 8, 12, 24

If you specify a multiple-expression for the day, week, month, quarter, or year date parts, IQ assumes the intervals started at the smallest date value (0000-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0000-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, then Sybase IQ calculates 10-day intervals starting at 0000-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a multiple that divides evenly into the next larger unit of time granularity.

If IQ rounds to a multiple of the week date part, the date value is always Sunday.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere

DATEFORMAT function [Date and time]

Function

Returns a string representing a date expression in the specified format.

Syntax

DATEFORMAT (*datetime-expression*, *string-expression*)

Parameters

datetime-expression The date/time to be converted. Must be a date, time, timestamp, or character string.

string-expression The format of the converted date.

Example

The following statement returns string values like “Jan 01, 1989”:

```
SELECT DATEFORMAT( start_date, 'Mmm dd, yyyy' ) from
Employees;
```

The following statement returns the string “Feb 19, 1987”:

```
SELECT DATEFORMAT( CAST ( ‘1987/02/19’ AS DATE ), ‘Mmm
Dd, yyyy’ ) FROM iq_dummy
```

Usage

The *datetime-expression* to convert must be a date, time, or timestamp data type, but can also be a CHAR or VARCHAR character string. If the date is a character string, Sybase IQ implicitly converts the character string to date, time, or timestamp data type, so an explicit cast, as in the example above, is unnecessary.

Any allowable date format can be used for *string-expression*. Date format strings cannot contain any multibyte characters. Only single-byte characters are allowed in a date/time/datetime format string, even when the collation order of the database is a multibyte collation order like 932JPN.

If '?' represents a multibyte character, then the following query fails:

```
SELECT DATEFORMAT ( start_date, ‘yy?’) FROM Employees;
```

Instead, move the multibyte character outside of the date format string using the concatenation operator:

```
SELECT DATEFORMAT (start_date, ‘yy’) + ‘?’ FROM
Employees;
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with SQL Anywhere

See also

DATE_FORMAT option in *Reference: Statements and Options*

DATENAME function [Date and time]

Function

Returns the name of the specified part (such as the month “June”) of a date/time value, as a character string.

Syntax

DATENAME (*date-part*, *date-expression*)

Parameters

date-part The date part to be named.

For a complete listing of allowed date parts, see “Date parts” on page 112.

date-expression The date for which the date part name is to be returned. The date must contain the requested *date-part*.

Example

The following statement returns the value May:

```
SELECT datename( month , '1987/05/02' ) FROM iq_dummy
```

Usage

DATENAME returns a character string, even if the result is numeric, such as 23, for the day.

Standards and compatibility

- **SQL92** Transact-SQL extension.

- **Sybase** Compatible with Adaptive Server Enterprise.

DATEPART function [Date and time]

Function
Returns an integer value for the specified part of a date/time value.

Syntax
DATEPART (*date-part*, *date-expression*)

Parameters
date-part The date part to be returned.

For a complete listing of allowed date parts, see “Date parts” on page 112.

date-expression The date for which the part is to be returned. The date must contain the *date-part* field.

Example
The following statement returns the value 5:

```
SELECT DATEPART( month , '1987/05/02' ) FROM iq_dummy
```

Usage
Note that the DATE, TIME, and DTTM indexes do not support some date parts (Calyearofweek, Calweekofyear, Caldayofweek, Dayofyear, Millisecond).

Standards and compatibility
- **SQL92** Transact-SQL extension.
- **Sybase** Compatible with Adaptive Server Enterprise.

DATEROUND function [Date and time]

Function
Calculates a new date, time, or datetime value by rounding the provided value up or down to the nearest multiple of the specified value with the specified granularity.

Syntax
DATEROUND (*date-part*, *datetime-expression [,multiple-expression]*)

Parameters
date part The date part to be added to the date.

The following date parts are not compatible with DATEROUND:

- DayofYear
- WeekDay
- CalYearofWeek
- CalWeekofYear
- CalDayofWeek

For a complete listing of date parts, see Date part values on page 113.

datetime-expression The date, time, or date-time expression containing the value you are evaluating.

multiple-expression (Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by the date_part parameter to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 10-minute intervals. Note that if multiple-expression evaluates to zero, evaluates to a negative number, or is an explicit NULL constant, then IQ generates an error. If multiple-expression evaluates to a NULL, then the function result is NULL.

Example

The following statement returns the value August 13, 2009, 10:30.000AM:

```
SELECT DATEROUND( MI, 'August 13, 2009 10:33.123AM', 10)
FROM iq_dummy
```

Usage

This function calculates a new date, time, or datetime value by rounding the provided value up or down to the nearest value with the specified granularity. If you include the optional multiple-expression parameter, then the function rounds the date and time to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the multiple-expression parameter.

If you specify a multiple-expression for the millisecond, second, minute, or hour date parts, IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day

For example, if you specify a multiple of two minutes, IQ applies two minute intervals starting at the current hour.

For the millisecond, second, minute, and hour date parts, specify a multiple-expression value that divides evenly into the range of the specified date part:

- For milliseconds, the valid multiple-expression values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For seconds and minutes, the valid multiple-expression values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60

- For hours, the valid multiple-expression values are: 1, 2, 3, 4, 6, 8, 12, 24

If you specify a multiple-expression for the day, week, month, quarter, or year date parts, IQ assumes the intervals started at the smallest date value (0000-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0000-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, then Sybase IQ calculates 10-day intervals starting at 0000-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a multiple that divides evenly into the next larger unit of time granularity.

If IQ rounds to a multiple of the week date part, then the date value is always Sunday.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere

DATETIME function [Date and time]

Function

Converts an expression into a timestamp.

Syntax

DATETIME (*expression*)

Parameters

expression The *expression* to be converted. The expression is usually a string. Conversion errors may be reported.

Example

The following statement returns a timestamp with value 1998-09-09 12:12:12.000:

```
SELECT DATETIME( '1998-09-09 12:12:12.000' ) FROM
iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

DAY function [Date and time]

Function

Returns an integer from 1 to 31 corresponding to the day of the month of the date specified.

Syntax

DAY (*date-expression*)

Parameters

date-expression The date.

Example

The following statement returns the value 12:

```
SELECT DAY( '2001-09-12' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise.

DAYNAME function [Date and time]

Function

Returns the name of the day of the week from the specified date.

Syntax

DAYNAME (*date-expression*)

Parameters

date-expression The date.

Example

The following statement returns the value Saturday:

```
SELECT DAYNAME ( '1987/05/02' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

DAYS function [Date and time]

Function

Returns the number of days since an arbitrary starting date, returns the number of days between two specified dates, or adds the specified *integer-expression* number of days to a given date.

Syntax

DAYS (*datetime-expression*)
| (*datetime-expression*, *datetime-expression*)
| (*datetime-expression*, *integer-expression*)

Parameters

datetime-expression A date and time.

integer-expression The number of days to be added to the *datetime-expression.* If the *integer-expression* is negative, the appropriate number of days are subtracted from the date/time. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a date.

For information on casting data types, see "CAST function [Data type conversion]" on page 133.

DAYS ignores hours, minutes, and seconds.

Examples

The following statement returns the integer value 729948:

```
SELECT DAYS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the integer value -366, which is the difference between the two dates:

```
SELECT DAYS( '1998-07-13 06:07:12',
'1997-07-12 10:07:12' ) FROM iq_dummy
```

The following statement returns the value 1999-07-14:

```
SELECT DAYS( CAST('1998-07-13' AS DATE ), 366 )
FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

DB_ID function [System]

Function

Returns the database ID number.

Syntax

DB_ID ([*database-name*])

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

database-name A string expression containing the database name. If *database-name* is a string constant, it must be enclosed in quotes. If no *database-name* is supplied, the ID number of the current database is returned.

Examples

The following statement returns the value 0, if iqdemo is the only running database:

```
SELECT DB_ID( 'iqdemo' ) FROM iq_dummy
```

The following statement returns the value 0, if executed against the only running database:

```
SELECT DB_ID() FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ

See also

"DB_NAME function [System]" on page 162

"OBJECT_ID function [System]" on page 215

DB_NAME function [System]

Function

Returns the database name.

Syntax

DB_NAME ([*database-id*])

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

database-id The ID of the database. *database-id* must be a numeric expression.

Example

The following statement returns the database name iqdemo, when executed against the sample database:

```
SELECT DB_NAME( 0 ) FROM iq_dummy
```

Usage

If no *database-id* is supplied, the name of the current database is returned.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ

See also

"COL_NAME function [System]" on page 138

"DB_ID function [System]" on page 161

"OBJECT_NAME function [System]" on page 216

DB_PROPERTY function [System]

Function

Returns the value of the given property.

Syntax

DB_PROPERTY ({ *property-id* | *property-name* }
[, { *database-id* | *database-name* }])

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

property-id The database property ID.

property-name The database property name.

database-id The database ID number, as returned by DB_ID. Typically, the database name is used.

database-name The name of the database, as returned by DB_NAME.

Example

The following statement returns the page size of the current database, in bytes.

```
SELECT DB_PROPERTY( 'PAGESIZE' ) FROM iq_dummy
```

Usage

Returns a string. The current database is used if the second argument is omitted.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"Properties available for each database" on page 122

"DB_ID function [System]" on page 161

"DB_NAME function [System]" on page 162

DEGREES function [Numeric]

Function

Converts a number from radians to degrees.

Syntax

DEGREES (*numeric-expression*)

Parameters

numeric-expression An angle in radians.

Example

The following statement returns the value 29.793805:

```
SELECT DEGREES( 0.52 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

DENSE_RANK function [Analytical]

Function

Ranks items in a group.

Syntax

DENSE_RANK () **OVER** (**ORDER BY** *expression* [**ASC** | **DESC**])

Parameters

expression A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Example

The following statement illustrates the use of the DENSE_RANK function:

```
SELECT s_suppkey, DENSE_RANK()
OVER ( ORDER BY ( SUM(s_acctBal) DESC )
AS rank_dense FROM supplier GROUP BY s_suppkey;
```

```
s_suppkey        sum_acctBal       rank_dense
supplier#011     200000            1
supplier#002     200000            1
supplier#013     123000            2
supplier#004     110000            3
supplier#035     110000            3
supplier#006     50000             4
supplier#021     10000             5
```

Usage

DENSE_RANK is a rank analytical function. The dense rank of row R is defined as the number of rows preceding and including R that are distinct within the groups specified in the OVER clause or distinct over the entire result set. The difference between DENSE_RANK and RANK is that DENSE_RANK leaves no gap in the ranking sequence when there is a tie. RANK leaves a gap when there is a tie.

DENSE_RANK requires an OVER (ORDER BY) clause. The ORDER BY clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the OVER clause and is *not* an ORDER BY for the SELECT. No aggregation functions in the rank query are allowed to specify DISTINCT.

The OVER clause indicates that the function operates on a query result set. The result set is the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses have all been evaluated. The OVER clause defines the data set of the rows to include in the computation of the rank analytical function.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

DENSE_RANK is allowed only in the select list of a SELECT or INSERT statement or in the ORDER BY clause of the SELECT statement. DENSE_RANK can be in a view or a union. The DENSE_RANK function cannot be used in a subquery, a HAVING clause, or in the select list of an UPDATE or DELETE statement. Only one rank analytical function is allowed per query.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere

See also

“Analytical functions” on page 104

“RANK function [Analytical]” on page 229

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

DIFFERENCE function [String]

Function

Compares two strings, evaluates the similarity between them, and returns a value from 0 to 4. The best match is 4.

Syntax

DIFFERENCE (*string-expression1*, *string-expression2*)

Parameters

string-expression1 The first string to compare.

string-expression2 The second string to compare.

Examples

The following statement returns the value 4:

```
SELECT DIFFERENCE( 'Smith', 'Smith' ) FROM iq_dummy
```

The following statement returns the value 4:

```
SELECT DIFFERENCE( 'Smith', 'Smyth' ) FROM iq_dummy
```

The following statement returns the value 3:

```
SELECT DIFFERENCE( 'Smith', 'Sweeney' ) FROM iq_dummy
```

The following statement returns the value 2:

```
SELECT DIFFERENCE( 'Smith', 'Jones' ) FROM iq_dummy
```

The following statement returns the value 1:

```
SELECT DIFFERENCE( 'Smith', 'Rubin' ) FROM iq_dummy
```

The following statement returns the value 0:

```
SELECT DIFFERENCE( 'Smith', 'Wilkins' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

"SOUNDEX function [String]" on page 255

DOW function [Date and time]

Function

Returns a number from 1 to 7 representing the day of the week of the specified date, with Sunday=1, Monday=2, and so on.

Syntax

DOW (*date-expression*)

Parameters

date-expression The date.

Example

The following statement returns the value 5:

```
SELECT DOW( '1998-07-09' ) FROM iq_dummy
```

Usage

See DATE_FIRST_DAY_OF_WEEK option in *Reference: Statements and Options* if you need Monday (or another day) to be the first day of the week.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

ERRORMSG function [Miscellaneous]

Function

Provides the error message for the current error, or for a specified SQLSTATE or SQLCODE value.

Syntax

ERRORMSG ([*sqlstate* | *sqlcode*] **)**

sqlstate: *string*

sqlcode: *integer*

Parameters

sqlstate The SQLSTATE value for which the error message is to be returned.

sqlcode The SQLCODE value for which the error message is to be returned.

Example

The following statement returns the error message for SQLCODE -813:

```
select errormsg( -813 )
```

Return value

A string containing the error message. If no argument is supplied, the error message for the current state is supplied. Any substitutions (such as table names and column names) are made.

If an argument is supplied, the error message for the supplied SQLSTATE or SQLCODE is returned, with no substitutions. Table names and column names are supplied as placeholders ('???').

The ERRORMSG function returns SQL Anywhere and Sybase IQ error messages.

Standards and compatibility

- **SQL92** Vendor extension
- **SQL99** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

System Administration Guide: Volume 2

EVENT_CONDITION function [System]

Function

To specify when an event handler is triggered.

Syntax

EVENT_CONDITION (*condition-name*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

condition-name The condition triggering the event. The possible values are preset in the database, and are case-insensitive. Each condition is valid only for certain event types. Table 4-20 lists the conditions and the events for which they are valid.

Table 4-20: Valid conditions for events

Condition name	Units	Valid for	Comment
DBFreePercent	N/A	DBDiskSpace	DBDiskSpace shows free space in the system database file (.db file), not the IQ store.
DBFreeSpace	Megabytes	DBDiskSpace	
DBSize	Megabytes	GrowDB	
ErrorNumber	N/A	RAISERROR	
IdleTime	Seconds	ServerIdle	
Interval	Seconds	All	Time since handler last executed.
LogFreePercent	N/A	LogDiskSpace	
LogFreeSpace	Megabytes	LogDiskSpace	
LogSize	Megabytes	GrowLog	
RemainingValues	Integer	GlobalAutoincrement	The number of remaining values.
TempFreePercent	N/A	TempDiskSpace	TempDiskSpace shows free space in the system temporary file (pointed to by TEMP or IQTMP15 environment variable), not the IQ temporary store.
TempFreeSpace	Megabytes	TempDiskSpace	
TempSize	Megabytes	GrowTemp	

Example

The following event definition uses the EVENT_CONDITION function:

```
create event LogNotifier
type LogDiskSpace
where event_condition( 'LogFreePercent' ) < 50
handler
begin
   message 'LogNotifier message'
end
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

CREATE EVENT statement in *Reference: Statements and Options*

EVENT_CONDITION_NAME function [System]

Function

Can be used to list the possible parameters for EVENT_CONDITION.

Syntax

EVENT_CONDITION_NAME (*integer*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

integer Must be greater than or equal to zero.

Usage

You can use EVENT_CONDITION_NAME to obtain a list of all EVENT_CONDITION arguments by looping over integers until the function returns NULL.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

CREATE EVENT statement in *Reference: Statements and Options*

EVENT_PARAMETER function [System]

Function

Provides context information for event handlers.

Syntax

EVENT_PARAMETER (*context-name*)

context-name:
'ConnectionID'
| **'User'**
| **'EventName'**

| 'Executions'
| 'IQDBMainSpaceName'
| 'NumActive'
| 'TableName'
| *condition-name*

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

context-name One of the preset strings. The strings are case-insensitive, and carry the following information:

- **ConnectionId** The connection ID, as returned by

```
connection_property( 'id' )
```

- **User** The user ID for the user that caused the event to be triggered.
- **EventName** The name of the event that has been triggered.
- **Executions** The number of times the event handler has been executed.
- **NumActive** The number of active instances of an event handler. This is useful if you want to limit an event handler so that only one instance executes at any given time.
- **TableName** The name of the table, for use with RemainingValues.

In addition, you can access any of the valid *condition-name* arguments to the EVENT_CONDITION function from the EVENT_PARAMETER function.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"EVENT_CONDITION function [System]" on page 166

CREATE EVENT statement in *Reference: Statements and Options*

EXP function [Numeric]

Function

Returns the exponential function, e to the power of a number.

Syntax

EXP (*numeric-expression*)

Parameters

numeric-expression The exponent.

Example

The following statement returns the value 3269017.3724721107:

```
SELECT EXP( 15 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

EXP_WEIGHTED_AVG function [Aggregate]

Function

Calculates an exponential weighted moving average. Weightings determine the relative importance of each quantity that makes up the average.

Syntax

EXP_WEIGHTED_AVG (*expression*, *period-expression*)
OVER (*window-spec*)

window-spec: See the Usage section, below.

Parameters

expression A numeric expression for which a weighted value is being computed.

period-expression A numeric expression specifying the period for which the average is to be computed.

Usage

Similar to the WEIGHTED_AVG function, the weights in EXP_WEIGHTED_AVG decrease over time. However, weights in WEIGHTED_AVG decrease arithmetically, whereas weights in EXP_WEIGHTED_AVG decrease exponentially. Exponential weighting applies more weight to the most recent values, and decreases the weight for older values while still applying some weight.

Sybase IQ calculates the exponential moving average using:

S*C+(1-S)*PEMA

In the calculation above, IQ applies the smoothing factor by multiplying the current closing price (C) by the smoothing constant (S) added to the product of the previous day's exponential moving average value (PEMA) and 1 minus the smoothing factor.

Sybase IQ calculates the exponential moving average over the entire period specified by the OVER clause. *period-expression* specifies the moving range of the exponential moving average.

You can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. The *window-spec* must contain an ORDER BY statement and cannot contain a frame specification. For information on how to specify the window, see "Analytical functions" on page 104.

Note ROLLUP and CUBE are not supported in the GROUP BY clause. DISTINCT is not supported.

Example

The following example returns an exponential weighted average of salaries for employees in Florida with the salary of recently hired employees contributing the most weight to the average. There are three rows used in the weighting:

```
SELECT DepartmentID, Surname, Salary,

EXP_WEIGHTED_AVG(Salary, 3) OVER (ORDER BY
YEAR(StartDate) DESC) as "W_AVG"

FROM Employees

WHERE State IN ('FL') ORDER BY StartDate DESC
```

The returned result set is:

Table 4-21: EXP_WEIGHTED_AVG result set

DepartmentID	Surname	Salary	W_AVG
400	Evans	68940.000	34470.000000
300	Litton	58930.000	46700.000000
200	Sterling	64900.000	55800.000000
200	Kelly	87500.000	71650.000000
400	Charlton	28300.000	49975.000000
100	Lull	87900.000	68937.500000
100	Gowda	59840.000	60621.875000
400	Francis	53870.000	61403.750000

Standards and compatibility

- **SQL2008** Vendor extension

FIRST_VALUE function [Aggregate]

Function

Returns the first value from a set of values.

Syntax

FIRST_VALUE (*expression* [IGNORE NULLS | RESPECT NULLS])

OVER (*window-spec*)

Parameters

expression The expression on which to determine the first value in an ordered set.

Usage

FIRST_VALUE returns the first value in a set of values, which is usually an ordered set. If the first value in the set is null, then the function returns NULL unless you specify IGNORE NULLS. If you specify IGNORE NULLS, then FIRST_VALUE returns the first non-null value in the set, or NULL if all values are null.

The data type of the returned value is the same as that of the input value.

You cannot use FIRST_VALUE or any other analytic function for *expression*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *expression*.

If the *window-spec* does not contain an ORDER BY, then the result is arbitrary. If there is no *window-spec*, the answer is arbitrary.

You can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Note DISTINCT is not supported.

Example

The following example returns the relationship, expressed as a percentage, between each employee’s salary and that of the most recently hired employee in the same department:

```
SELECT DepartmentID, EmployeeID,
100 * Salary / ( FIRST_VALUE( Salary ) OVER (
PARTITION BY DepartmentID  ORDER BY Year(StartDate) DESC
) )
AS percentage
FROM Employees order by DepartmentID DESC;
```

The returned result set is:

Table 4-22: FIRST_VALUE result set

DepartmentID	EmployeeID	Percentage
500	1658	100.00000000000000000000
500	1570	138.842709713689113761394
500	1615	110.428462434244870095972
500	1013	109.585190539292454724330
500	750	137.734409508894510701521
500	921	167.449704854836766654619
500	868	113.239368750752921334778
500	703	222.867927558928643135365
500	191	119.664297474199895594908
400	1684	100.00000000000000000000
400	1740	76.128652163477274215016
400	1751	76.353400685155687446813
400	1607	133.758100765890593292456
400	1507	77.996465120338650199655
400	1576	150.428767810774836893669

In this example, employee 1658 is the first row for department 500, indicating that employee 1658 is the most recent hire in that department, and therefore receives a percentage of 100%. Percentages for the remaining employees in department 500 are calculated relative to that of employee 1658. For example, employee 1570 earns approximately 139% of what employee 1658 earns.

Standards and compatibility

- **SQL2008** SQL/OLAP feature T612
- **Sybase** Compatible with SQL Anywhere

FLOOR function [Numeric]

Function

Returns the floor of (largest integer not greater than) a number.

Syntax

FLOOR (*numeric-expression*)

Parameters

numeric-expression The number, usually a float.

Examples

The following statement returns the value 123.00:

```
SELECT FLOOR ( 123 ) FROM iq_dummy
```

The following statement returns the value 123:

```
SELECT FLOOR ( 123.45 ) FROM iq_dummy
```

The following statement returns the value -124.00.

```
SELECT FLOOR ( -123.45 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

"CEILING function [Numeric]" on page 135

GETDATE function [Date and time]

Function

Returns the current date and time.

Syntax

GETDATE ()

Example

The following statement returns the system date and time.

```
SELECT GETDATE( ) FROM iq_dummy
```

Usage

GETDATE is a Transact-SQL compatible data manipulation function.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

GRAPHICAL_PLAN function [String]

Function

Returns the graphical query plan to Interactive SQL in an XML format string.

Syntax

GRAPHICAL_PLAN (*string-expression*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

string-expression SQL statement for which the plan is to be generated. *string-expression* is generally a SELECT statement, but it can also be an UPDATE or DELETE, INSERT SELECT, or SELECT INTO statement.

If you do not provide an argument to the GRAPHICAL_PLAN function, the query plan is returned to you from the cache. If there is no query plan in the cache, then this message appears:

```
plan not available
```

The behavior of GRAPHICAL_PLAN function is controlled by database options QUERY_PLAN_TEXT_ACCESS and QUERY_PLAN_TEXT_CACHING. If QUERY_PLAN_TEXT_ACCESS is OFF (the default), then this message appears:

```
Plan not available. The database option
QUERY_PLAN_TEXT_ACCESS is OFF
```

If a user needs access to the plan, the DBA must set option QUERY_PLAN_TEXT_ACCESS ON for that user.

If QUERY_PLAN_TEXT_ACCESS is ON, and the query plan for the string expression is available in the cache maintained on the server, the query plan from the cache is returned to you.

If the query plan is not available in the cache and you are authorized to view plans on the client, then a query plan with optimizer estimates (query plan with NOEXEC option ON) is generated and appears on the dbisql client plan window.

Note Sybase IQ does not support NOEXEC plan generation for SELECT, UPDATE, DELETE, INSERT SELECT, and SELECT INTO queries.

When a user requests a query plan that has not yet been executed, the query plan is not available in the cache. Instead, a query plan with optimizer estimates is returned without QUERY_PLAN_AFTER_RUN statistics.

Query plans for stored procedures are not accessible using the GRAPHICAL_PLAN function.

Users can view the query plan for cursors opened for IQ queries. A cursor is declared and opened using DECLARE CURSOR and OPEN CURSOR commands. To obtain the query plan for the most recently opened cursor, use:

```
SELECT GRAPHICAL_PLAN ( );
```

With the QUERY_PLAN_AFTER_RUN option OFF, the plan appears after OPEN CURSOR or CLOSE CURSOR. However, if QUERY_PLAN_AFTER_RUN is ON, CLOSE CURSOR must be executed before you request the plan.

For information on viewing the query optimizer's execution plan for a SQL statement in the Plan Viewer window in Interactive SQL, see “Viewing graphical plans in Interactive SQL” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Administering Your Database > SQL Anywhere graphical administration tools > Using Interactive SQL > Viewing plans using the Interactive SQL Plan Viewer.*

When Interactive SQL users request plans for UPDATE, DELETE, SELECT INTO, and INSERT SELECT queries, the NOEXEC plan is not supported. To access the query plan, first explicitly execute the query, and then request the plan.

Examples

The following example passes a SELECT statement as a string parameter and returns the plan for executing the query. It saves the plan in the file *gplan.xml*.

Note If you use the OUTPUT statement’s HEXADECIMAL clause set to ASIS to get formatted plan output, the values of characters are written without any escaping, even if the value contains control characters. ASIS is useful for text that contains formatting characters such as tabs or carriage returns.

```
SELECT GRAPHICAL_PLAN ('SELECT * FROM
Employees');OUTPUT to 'C:\gplan.xml' HEXADECIMAL ASIS
quote '';
```

The following example returns the query plan from the cache, if available:

```
SELECT GRAPHICAL_PLAN ( );
```

Standards and compatibility

- **SQL92** Vendor extension
- **SQL99** SQL/foundation feature outside of core SQL
- **Sybase** Not supported by Adaptive Server Enterprise

See also

“HTML_PLAN function [String]” on page 183

“OUTPUT statement [DBISQL]” in *Reference: Building Blocks, Tables, and Procedures*

“NOEXEC option,” “QUERY_PLAN_AFTER_RUN option,” “QUERY_PLAN_AS_HTML option,” “QUERY_PLAN_TEXT_ACCESS option,”and “QUERY_PLAN_TEXT_CACHING option” in *Reference: Statements and Options*

PLAN function [Miscellaneous], EXPLANATION function, [Miscellaneous], GRAPHICAL_ULPLAN function [Miscellaneous], LONG_ULPLAN function [Miscellaneous], and SHORT_ULPLAN function [Miscellaneous] in *SQL Anywhere Server – SQL Reference*.

GROUPING function [Aggregate]

Function

Identifies whether a column in a ROLLUP or CUBE operation result set is NULL because it is part of a subtotal row, or NULL because of the underlying data.

Syntax

GROUPING (*group-by-expression*)

Parameters

group-by-expression An expression appearing as a grouping column in the result set of a query that uses a GROUP BY clause with the ROLLUP or CUBE keyword. The function identifies subtotal rows added to the result set by a ROLLUP or CUBE operation.

Currently, Sybase IQ does not support the PERCENTILE_CONT or PERCENTILE_DISC functions with GROUP BY CUBE operations.

Return value

- **1** Indicates that *group-by-expression* is NULL because it is part of a subtotal row. The column is not a prefix column for that row.
- **0** Indicates that *group-by-expression* is a prefix column of a subtotal row.

Standards and compatibility

- **SQL92** Vendor extension
- **SQL99** SQL/foundation feature outside of core SQL
- **Sybase** Not supported by Adaptive Server Enterprise

See also

SELECT statement in *Reference: Statements and Options*

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

GROUP_MEMBER function [System]

Function

Identifies whether the user belongs to the specified group.

Syntax

GROUP_MEMBER (*group-name-string-expression*[, *user-name-string-expression*])

Parameters

group-name-string-expression Identifies the group to be considered.

user-name-string-expression Identifies the user to be considered. If not supplied, then the current user name is assumed.

Return value

- **0** Returns 0 if the group does not exist, if the user does not exist, or if the user does not belong to the specified group.
- **1** Returns an integer other than 0 if the user is a member of the specified group.

Standards and compatibility

- **SQL92** Vendor extension
- **SQL99** SQL/foundation feature outside of core SQL
- **Sybase** Not supported by Adaptive Server Enterprise

HEXTOBIGINT function [Data type conversion]

Function

Returns the BIGINT equivalent of a hexadecimal string.

Syntax

HEXTOBIGINT (*hexadecimal-string*)

Parameters

hexadecimal-string The hexadecimal value to be converted to a big integer (BIGINT). Input can be in the following forms, with either a lowercase or uppercase “0x” in the prefix, or no prefix:

```
0xhex-string
0Xhex-string
hex-string
```

Examples

The following statements return the value 4294967287:

```
SELECT HEXTOBIGINT ( '0xfffffff7' ) FROM iq_dummy
SELECT HEXTOBIGINT ( '0Xfffffff7' ) FROM iq_dummy
SELECT HEXTOBIGINT ( 'fffffff7' ) FROM iq_dummy
```

Usage

The HEXTOBIGINT function accepts hexadecimal integers and returns the BIGINT equivalent. Hexadecimal integers can be provided as CHAR and VARCHAR value expressions, as well as BINARY and VARBINARY expressions.

The HEXTOBIGINT function accepts a valid hexadecimal string, with or without a “0x” or “0X” prefix, enclosed in single quotes.

Input of fewer than 16 digits is assumed to be left-padded with zeros.

For data type conversion failure on input, Sybase IQ returns an error unless the CONVERSION_ERROR option is set to OFF. When CONVERSION_ERROR is OFF, invalid hexadecimal input returns NULL.

An error is returned if a BINARY or VARBINARY value exceeds 8 bytes and a CHAR or VARCHAR value exceeds 16 characters, with the exception of the value being appended with ‘0x.’

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

CONVERSION_ERROR option [TSQL] in *Reference: Statements and Options*

"BIGINTTOHEX function [Data type conversion]" on page 132

"HEXTOINT function [Data type conversion]" on page 179

"INTTOHEX function [Data type conversion]" on page 189

HEXTOINT function [Data type conversion]

Function

Returns the unsigned BIGINT equivalent of a hexadecimal string.

Syntax

HEXTOINT (*hexadecimal-string*)

Parameters

hexadecimal-string The string to be converted to an integer. Input can be in the following forms, with either a lowercase or uppercase "x" in the prefix, or no prefix:

```
0xhex-string
0Xhex-string
hex-string
```

Examples

The following statements return the value 420:

```
SELECT HEXTOINT ( '0x1A4' ) FROM iq_dummy
SELECT HEXTOINT ( '0X1A4' ) FROM iq_dummy
SELECT HEXTOINT ( '1A4' ) FROM iq_dummy
```

Usage

For invalid hexadecimal input, Sybase IQ returns an error unless the CONVERSION_ERROR option is OFF. When CONVERSION_ERROR is OFF, invalid hexadecimal input returns NULL.

The database option ASE_FUNCTION_BEHAVIOR specifies that output of Sybase IQ functions, including INTTOHEX and HEXTOINT, is consistent with the output of Adaptive Server Enterprise functions. When the ASE_FUNCTION_BEHAVIOR option is ON:

- Sybase IQ HEXTOINT assumes input is a hexadecimal string of 8 characters; if the length is less than 8 characters long, the string is left padded with zeros.
- Sybase IQ HEXTOINT accepts a maximum of 16 characters prefixed with 0x (a total of 18 characters); use caution, as a large input value can result in an integer value that overflows the 32-bit signed integer output size.

- The data type of the output of the Sybase IQ HEXTOINT function is assumed to be a 32-bit signed integer.
- Sybase IQ HEXTOINT accepts a 32-bit hexadecimal integer as a signed representation.
- For more than 8 hexadecimal characters, Sybase IQ HEXTOINT considers only relevant characters.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

ASE_FUNCTION_BEHAVIOR option in *Reference: Statements and Options*

CONVERSION_ERROR option [TSQL] in *Reference: Statements and Options*

"INTTOHEX function [Data type conversion]" on page 189

HOUR function [Date and time]

Function

Returns a number from 0 to 23 corresponding to the hour component of the specified date/time.

Syntax

HOUR (*datetime-expression*)

Parameters

datetime-expression The date/time.

Example

The following statement returns the value 21:

```
SELECT HOUR( '1998-07-09 21:12:13' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

HOURS function [Date and time]

Function

Returns the number of hours since an arbitrary starting date and time, the number of whole hours between two specified times, or adds the specified integer-expression number of hours to a time.

Syntax

HOURS (*datetime-expression*
| *datetime-expression*, *datetime-expression*
| *datetime-expression*, *integer-expression*)

Parameters

datetime-expression A date and time.

integer-expression The number of hours to be added to the *datetime-expression.* If *integer-expression* is negative, the appropriate number of hours are subtracted from the date/time. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a datetime data type.

For information on casting data types, see "CAST function [Data type conversion]" on page 133.

Examples

The following statement returns the value 17518758:

```
SELECT HOURS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 4, to signify the difference between the two times:

```
SELECT HOURS( '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-05-13 02:05:07.000:

```
SELECT HOURS( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5 ) FROM iq_dummy
```

Usage

The second syntax returns the number of whole hours from the first date/time to the second date/time. The number might be negative.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

HTML_DECODE function [HTTP]

Function

Decodes special character entities that appear in HTML literal strings.

Syntax

HTML_DECODE (*string*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

string An arbitrary literal string used in an HTML document.

Usage

This function returns the string argument after making the following set of substitutions:

Characters	Substitution
"	"

Characters	Substitution
'	'
&	&
<	<
>	>
&#xnn;	character nn

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"HTML_ENCODE function [HTTP]" on page 182

"HTTP_ENCODE function [HTTP]" on page 185

HTML_ENCODE function [HTTP]

Function

Encodes special characters within strings to be inserted into HTML documents.

Syntax

HTML_ENCODE (*string*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

string An arbitrary literal string used in an HTML document.

Usage

This function returns the string argument after making the following set of substitutions:

Characters	Substitution
"	"
'	'
&	&
<	<
>	>
codes no less than 0X20	&#xnn

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"HTML_DECODE function [HTTP]" on page 181

"HTTP_ENCODE function [HTTP]" on page 185

HTML_PLAN function [String]

Function

Returns query plans in an HTML format string.

Syntax

HTML_PLAN (*string-expression*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

string-expression SQL statement for which the plan is to be generated. It is primarily a SELECT statement but can be an UPDATE or DELETE statement.

If you do not provide an argument to the HTML_PLAN function, the query plan is returned to you from the cache. If there is no query plan in the cache, this message appears:

```
No plan available
```

The behavior of the HTML_PLAN function is controlled by database options QUERY_PLAN_TEXT_ACCESS and QUERY_PLAN_TEXT_CACHING. If QUERY_PLAN_TEXT_ACCESS is OFF (the default), this message appears:

```
Plan not available. The database option
QUERY_PLAN_TEXT_ACCESS is OFF
```

If QUERY_PLAN_TEXT_ACCESS is ON, and the query plan for the string expression is available in the cache maintained on the server, the query plan from the cache is returned to you.

The HTML_PLAN function can be used to return query plans to Interactive SQL using SELECT, UPDATE, DELETE, INSERT SELECT, and SELECT INTO.

Users can view the query plan for cursors opened for IQ queries. To obtain the query plan for the most recently opened cursor, use:

```
SELECT HTML_PLAN ( );
```

With QUERY_PLAN_AFTER_RUN option OFF, the plan appears after OPEN CURSOR or CLOSE CURSOR. However, if QUERY_PLAN_AFTER_RUN is ON, CLOSE CURSOR must be executed before you request the plan.

For information on viewing the query optimizer's execution plan for a SQL statement in the Plan Viewer window in Interactive SQL, see "Viewing plans using the Interactive SQL Plan Viewer " in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Administering Your Database > SQL Anywhere graphical administration tools > Using Interactive SQL*.

When Interactive SQL users request plans for UPDATE, DELETE, SELECT INTO, and INSERT SELECT queries, the NOEXEC plan is not supported. To access the query plan, first explicitly execute the query, then request the plan.

When you request an HTML_PLAN for a SQL Anywhere query or for an OMNI/CIS decomposed query, the following message is returned:

```
No plan. HTML_PLAN function is not supported for this
type of statement or database.
```

Examples

The following example passes a SELECT statement as a string parameter and returns the HTML plan for executing the query. It saves the plan in the file *hplan.html*.

```
SELECT HTML_PLAN ('SELECT * FROM Employees');
OUTPUT to 'C:\hplan.html' HEXADECIMAL ASIS QUOTE '';
```

The OUTPUT TO clause HEXADECIMAL ASIS is useful for text that contains formatting characters such as tabs or carriage returns. When set to ASIS, values are written as is, without any escaping, even if the values contain control characters.

The following example returns the HTML query plan from the cache, if available.

```
SELECT HTML_PLAN ( );
```

Standards and compatibility

- **SQL92** Vendor extension
- **SQL99** SQL/foundation feature outside of core SQL
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"GRAPHICAL_PLAN function [String]" on page 174

"OUTPUT statement [DBISQL]" in *Reference: Building Blocks, Tables, and Procedures*

"NOEXEC option," "QUERY_PLAN_AFTER_RUN option," "QUERY_PLAN_AS_HTML option," "QUERY_PLAN_TEXT_ACCESS option,"and "QUERY_PLAN_TEXT_CACHING option" in *Reference: Statements and Options*

PLAN function [Miscellaneous], EXPLANATION function, [Miscellaneous], GRAPHICAL_ULPLAN function [Miscellaneous], LONG_ULPLAN function [Miscellaneous], and SHORT_ULPLAN function [Miscellaneous] in *SQL Anywhere Server – SQL Reference*.

HTTP_DECODE function [HTTP]

Function

Decodes special characters within strings for use with HTTP.

Syntax

HTTP_DECODE (*string*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

string Arbitrary string to be used in an HTTP request.

Usage

This function returns the string argument after replacing all character sequences of the form %*nn*, where *nn* is a hexadecimal value, with the character with code *nn*. In addition, all plus signs (+) are replaced with spaces.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"HTML_ENCODE function [HTTP]" on page 182

"HTTP_ENCODE function [HTTP]" on page 185

HTTP_ENCODE function [HTTP]

Function

Encodes special characters in strings for use with HTTP.

Syntax

HTML_ENCODE (*string*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

string Arbitrary string to be used in an HTTP request.

Usage

This function returns the string argument after making the following set of substitutions. In addition, all characters with hexadecimal codes less than 1F or greater than 7E are replaced with %*nn*, where *nn* is the character code.

Character	Substitution
space	%20
"	%22
#	%23
&	%26
,	%2C
;	%3B
<	%3C
>	%3E
[	%5B
\	%5C
]	%5D
`	%60
{	%7B
|	%7C
}	%7D

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"HTML_ENCODE function [HTTP]" on page 182

"HTTP_DECODE function [HTTP]" on page 185

HTTP_HEADER function [HTTP]

Function

Gets the value of an HTTP header.

Syntax

HTML_HEADER (*field-name*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

field-name The name of an HTTP header field.

Usage

This function returns the value of the named HTTP header field. It is used when processing an HTTP request through a Web service.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"HTTP_VARIABLE function [HTTP]" on page 187

"NEXT_HTTP_HEADER function [HTTP]" on page 210

"NEXT_HTTP_VARIABLE function [HTTP]" on page 211

HTTP_VARIABLE function [HTTP]

Function

Gets the value of an HTTP variable.

Syntax

HTML_VARIABLE (*var-name* [[, *instance*] , *header-field*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

var-name The name of the an HTTP variable.

instance If more than one variable has the same name, the instance number of the field instance, or NULL to get the first one. Useful for select lists that permit multiple selections.

header-field In a multipart request, a header field name associated with the named field.

Usage

This function returns the value of the named HTTP variable. It is used when processing an HTTP request through a Web service.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

"HTML_DECODE function [HTTP]" on page 181

"NEXT_HTTP_HEADER function [HTTP]" on page 210

"NEXT_HTTP_VARIABLE function [HTTP]" on page 211

IFNULL function [Miscellaneous]

Function

If the first expression is the NULL value, then the value of the second expression is returned. If the first expression is not NULL, the value of the third expression is returned. If the first expression is not NULL and there is no third expression, then the NULL value is returned.

Syntax

IFNULL (*expression1*, *expression2* [, *expression3*])

Parameters

expression1 The expression to be evaluated. Its value determines whether *expression2* or *expression3* is returned.

expression2 The return value if *expression1* is NULL.

expression3 The return value if *expression1* is not NULL.

Examples

The following statement returns the value -66:

```
SELECT IFNULL( NULL, -66 ) FROM iq_dummy
```

The following statement returns NULL, because the first expression is not NULL and there is no third expression:

```
SELECT IFNULL( -66, -66 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Transact-SQL extension
- **Sybase** Not supported by Adaptive Server Enterprise

INDEX_COL function [System]

Function

Returns the name of the indexed column.

Syntax

INDEX_COL (*table-name*, *index-id*, *key_#* [, *user-id*])

Parameters

table-name A table name.

index-id The index ID of an index of *table-name*.

key_# A key in the index specified by *index-id*. This parameter specifies the column number in the index. For a single column index, *key_#* is equal to 0. For a multicolumn index, *key_#* is equal to 0 for the first column, 1 for the second column, and so on.

user-id The user ID of the owner of *table-name*. If *user-id* is not specified, this value defaults to the caller's user ID.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ

See also

"OBJECT_ID function [System]" on page 215

INSERTSTR function [String]

Function

Inserts a string into another string at a specified position.

Syntax

INSERTSTR (*numeric-expression*, *string-expression1*, *string-expression2*)

Parameters

numeric-expression The position after which *string-expression2* is to be inserted. Use zero to insert a string at the beginning.

string-expression1 The string into which *string-expression2* is to be inserted.

string-expression2 The string to be inserted.

Note The result data type of an INSERTSTR function is a LONG VARCHAR. If you use INSERTSTR in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set INSERTSTR to the correct data type and size.

See “REPLACE function [String]” .

Example

The following statement returns the value “backoffice”:

```
SELECT INSERTSTR( 0, 'office ', 'back' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported in Adaptive Server Enterprise. The STUFF function is equivalent and is supported in both Adaptive Server Enterprise and Sybase IQ

See also

“STUFF function [String]” on page 264

INTTOHEX function [Data type conversion]

Function

Returns the hexadecimal equivalent of a decimal integer.

Syntax

INTTOHEX (*integer-expression*)

Parameters

integer-expression The integer to be converted to hexadecimal.

Examples

The following statement returns the value 3B9ACA00:

```
SELECT INTTOHEX( 1000000000 ) FROM iq_dummy
```

The following statement returns the value 00000002540BE400:

```
SELECT INTTOHEX ( 10000000000) FROM iq_dummy
```

Usage

If data conversion of input to INTTOHEX conversion fails, Sybase IQ returns an error, unless the CONVERSION_ERROR option is OFF. In that case, the result is NULL.

ASE_FUNCTION_BEHAVIOR option The database option ASE_FUNCTION_BEHAVIOR specifies that output of IQ functions, including INTTOHEX and HEXTOINT, be consistent with the output of Adaptive Server Enterprise functions. The default value of ASE_FUNCTION_BEHAVIOR is OFF.

When the ASE_FUNCTION_BEHAVIOR option is disabled (the value is OFF):

- The output of INTTOHEX is compatible with SQL Anywhere.
- Depending on the input, the output of INTTOHEX can be 8 digits or 16 digits and is left padded with zeros; the return data type is VARCHAR.
- The output of INTTOHEX does not have a ‘0x’ or ‘0X’ prefix.
- The input to INTTOHEX can be up to a 64-bit integer.

When the ASE_FUNCTION_BEHAVIOR option is enabled (the value is ON):

- The output of INTTOHEX is compatible with ASE.
- The output of INTTOHEX is always 8 digits and is left-padded with zeros; the return data type is VARCHAR.
- The output of INTTOHEX does not have a ‘0x’ or ‘0X’ prefix.
- Sybase IQ INTTOHEX assumes input is a 32-bit signed integer; a larger value can overflow and a conversion error can result. For example, the statement:

  ```
  SELECT INTTOHEX( 1000000000 ) FROM iq_dummy
  ```

  returns the value `3B9ACA00`. But the statement:

  ```
  SELECT INTTOHEX( 10000000000 ) FROM iq_dummy
  ```

  results in a conversion error.

Standards and compatibility

- **SQL92** Transact-SQL extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

CONVERSION_ERROR option [TSQL] in *Reference: Statements and Options*

ASE_FUNCTION_BEHAVIOR option in *Reference: Statements and Options*

“HEXTOINT function [Data type conversion]” on page 179

ISDATE function [Date and time]

Function
Tests whether a string argument can be converted to a date. If a conversion is possible, the function returns 1; otherwise, it returns 0. If the argument is null, 0 is returned.

Syntax
ISDATE (*string*)

Parameters
string The string to be analyzed to determine whether the string represents a valid date.

Example
The following example tests whether the birth_date column holds valid dates, returning invalid dates as NULL, and valid dates in date format.

```
select birth_date from MyData;
------------------------------
1990/32/89
0101/32/89
1990/12/09

select
  case when isdate(birth_date)=0 then NULL
  else cast(birth_date as date)
  end
  from MyData;
------------------------------------
(NULL)
(NULL)
1990-12-09
```

Standards and compatibility

- **SQL92** Vendor extension
- **SQL99** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

ISNULL function [Miscellaneous]

Function
Returns the value of the first non-NULL expression in the parameter list.

Syntax
ISNULL (*expression*, *expression* […, *expression*])

Parameters
expression An expression to be tested against NULL.

At least two expressions must be passed to the function.

Example
The following statement returns the value -66:

```
SELECT ISNULL( NULL ,-66, 55, 45, NULL, 16 ) FROM
iq_dummy
```

Usage — The ISNULL function is the same as the COALESCE function.

Standards and compatibility

- **SQL92** Transact-SQL extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also — "COALESCE function [Miscellaneous]" on page 137

ISNUMERIC function [Miscellaneous]

Function — Tests whether a string argument can be converted to a numeric. If a conversion is possible, the function returns 1; otherwise, it returns 0. If the argument is null, 0 is returned.

Syntax

ISNUMERIC (*string*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters — **string** The string to be analyzed to determine whether the string represents a valid numeric value.

Usage — For optimal performance, avoid using ISNUMERIC in predicates, where it is processed by the SQL Anywhere portion of the product and cannot take advantage of the performance features of Sybase IQ.

Example — The following example tests whether the height_in_cms column holds valid numeric data, returning invalid numeric data as NULL, and valid numeric data in int format.

```
data height_in_cms
------------------------
asde
asde
180
156
```

```
select case
   when isnumeric(height_in_cms)=0
   then NULL
   else cast(height_in_cms as int)
   end
from MyData
```

Standards and compatibility

- **SQL92** Vendor extension

- **SQL99** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

LAST_VALUE function [Aggregate]

Function

Returns the last value from a set of values.

Syntax

LAST_VALUE (*expression* [IGNORE NULLS | RESPECT NULLS])
OVER (*window-spec*)

Parameters

expression The expression on which to determine the last value in an ordered set.

Usage

LAST_VALUE returns the last value in a set of values, which is usually an ordered set. If the last value in the set is null, then the function returns NULL unless you specify IGNORE NULLS. If you specify IGNORE NULLS, then LAST_VALUE returns the last non-null value in the set, or NULL if all values are null.

The data type of the returned value is the same as that of the input value.

You cannot use LAST_VALUE or any other analytic function for *expression*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *expression*.

If the *window-spec* does not contain an ORDER BY then the result is arbitrary. If there is no *window-spec*, then the result is arbitrary.

You can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see "Analytical functions" on page 104.

Note DISTINCT is not supported.

Example

The following example returns the salary of each employee, plus the name of the employee with the highest salary in their department:

```
SELECT GivenName + ' ' + Surname AS employee_name,
   Salary, DepartmentID,
   LAST_VALUE( employee_name ) OVER Salary_Window AS
highest_paid
FROM Employees
WINDOW Salary_Window AS ( PARTITION BY DepartmentID
ORDER BY Salary
   RANGE BETWEEN UNBOUNDED PRECEDING
```

```
    AND UNBOUNDED FOLLOWING )
ORDER BY DepartmentID DESC;
```

The returned result set is:

Table 4-23:

employee_name	Salary	DepartmentID	highest_paid
Michael Lynch	24903.000	500	Jose Martinez
Joseph Barker	27290.000	500	Jose Martinez
Sheila Romero	27500.000	500	Jose Martinez
Felicia Kuo	28200.000	500	Jose Martinez
Jeannette Bertrand	29800.000	500	Jose Martinez
Jane Braun	34300.000	500	Jose Martinez
Anthony Rebeiro	34576.000	500	Jose Martinez
Charles Crowley	41700.000	500	Jose Martinez
Jose Martinez	55500.800	500	Jose Martinez
Doug Charlton	28300.000	400	Scott Evans
Elizabeth Lambert	29384.000	400	Scott Evans
Joyce Butterfield	34011.000	400	Scott Evans
Robert Nielsen	34889.000	400	Scott Evans
Alex Ahmed	34992.000	400	Scott Evans
Ruth Wetherby	35745.000	400	Scott Evans
...	...	...	...

Standards and compatibility

- **SQL2008** SQL/OLAP feature T612
- **Sybase** Compatible with SQL Anywhere

LCASE function [String]

Function

Converts all characters in a string to lowercase.

Syntax

LCASE (*string-expression*)

Parameters

string-expression The string to be converted to lowercase.

Note The result data type of an LCASE function is a LONG VARCHAR. If you use LCASE in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set LCASE to the correct data type and size.

See "REPLACE function [String]" for more information.

Example

The following statement returns the value "lower case":

```
SELECT LCASE( 'LOWER CasE' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** LCASE is not supported in Adaptive Server Enterprise; you can use LOWER to get the same functionality

See also

"LOWER function [String]" on page 200

"UCASE function [String]" on page 329

"UPPER function [String]" on page 329

LEFT function [String]

Function

Returns a specified number of characters from the beginning of a string.

Syntax

LEFT (*string-expression*, *numeric-expression*)

Parameters

string-expression The string.

numeric-expression The number of characters to return.

Example

The following statement returns the value "choco":

```
SELECT LEFT( 'chocolate', 5 ) FROM iq_dummy
```

Usage

If the string contains multibyte characters, and the proper collation is being used, the number of bytes returned may be greater than the specified number of characters.

Note The result data type of a LEFT function is a LONG VARCHAR. If you use LEFT in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set LEFT to the correct data type and size.

See "REPLACE function [String]" for more information.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"RIGHT function [String]" on page 245

Chapter 11, "International Languages and Character Sets" in the *System Administration Guide: Volume 1*

LEN function [String]

Function

Takes one argument as an input of type BINARY or STRING and returns the number of characters, as defined by the database's collation sequence, of a specified string expression, excluding trailing blanks. The result may differ from the string's byte length for multi-byte character sets.

BINARY and VARBINARY are also allowed, in which case LEN() returns the number of bytes of the input.

LEN is an alias of LENGTH function

Syntax

LEN (*string_expr*)

Parameters

string_expr is the string expression to be evaluated.

Example

The following example returns the value 3152:

```
select len(Photo) from Products

where ID = 500
```

Usage

This function is the equivalent of CHAR_LENGTH (*string_expression*).

Permissions

Any user can execute LEN.

Standards and compatibility

ANSI SQL – Compliance level: Transact-SQL extension

See also

Data types CHAR, NCHAR, VARCHAR, NVARCHAR.

See Chapter 3, “SQL Data Types.”

Functions “CHAR_LENGTH function [String]” on page 136 and “STR_REPLACE function [String]” on page 261.

For general information about string functions, see “String functions” on page 115.

LENGTH function [String]

Function

Returns the number of characters in the specified string.

Syntax

LENGTH (*string-expression*)

Parameters

string-expression The string.

Example

The following statement returns the value 9:

```
SELECT LENGTH( 'chocolate' ) FROM iq_dummy
```

Usage

If the string contains multibyte characters, and the proper collation is being used, LENGTH returns the number of characters, not the number of bytes. If the string is of BINARY data type, the LENGTH function behaves as BYTE_LENGTH.

The LENGTH function is the same as the CHAR_LENGTH function.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise. Use the CHAR_LENGTH function instead

See also

“BYTE_LENGTH function [String]” on page 133

“CHAR_LENGTH function [String]” on page 136

Chapter 11, “International Languages and Character Sets” in the *System Administration Guide: Volume 1*

LN function [Numeric]

Function

Returns the natural logarithm of the specified expression.

Syntax

LN (*numeric-expression*)

Parameters

expression Is a column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, the LN function generates an error. The return value is of DOUBLE data type.

Usage

LN takes one argument. For example, LN (*20*) returns 2.995732.

The LN function is an alias of the LOG function.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise. Use the LOG function instead

See also

"LOG function [Numeric]" on page 199

Chapter 11, "International Languages and Character Sets" in the *System Administration Guide: Volume 1*

LOCATE function [String]

Function

Returns the position of one string within another.

Syntax

LOCATE (*string-expression1*, *string-expression2*
[, *numeric-expression*])

Parameters

string-expression1 The string to be searched.

string-expression2 The string for which you are searching. This string is limited to 255 bytes.

numeric-expression The character position at which to begin the search in the string. The first character is position 1. If the starting offset is negative, LOCATE returns the last matching string offset, rather than the first. A negative offset indicates how much of the end of the string to exclude from the search. The number of bytes excluded is calculated as (-1 * offset) - 1.

The *numeric-expression* is a 32 bit signed integer for CHAR, VARCHAR, and BINARY columns.

Examples

The following statement returns the value 8:

```
SELECT LOCATE( 'office party this week - rsvp as soon
as possible', 'party', 2 ) FROM iq_dummy
```

In the second example, the *numeric-expression* starting offset for the search is a negative number.

```
CREATE TABLE t1(name VARCHAR(20), dirname VARCHAR(60));
  INSERT INTO t1
```

```
    VALUES('m1000','c:\test\functions\locate.sql');
  INSERT INTO t1
    VALUES('m1001','d:\test\functions\trim.sql');
COMMIT;

SELECT LOCATE(dirname, '\', -1), dirname FROM t1;
```

The result is:

```
18   c:\test\functions\locate.sql
18   d:\test\functions\trim.sql
```

Usage

If *numeric-expression* is specified, the search starts at that offset into the string being searched.

If *numeric-expression* is not specified, LOCATE returns only the position of the first instance of the specified string.

The first string can be a long string (longer than 255 bytes), but the second is limited to 255 bytes. If a long string is given as the second argument, the function returns a NULL value.

If any of the arguments is NULL, the result is NULL.

Searching for a zero-length string returns 1.

If the string does not contain the specified string, the LOCATE function returns zero (0).

All the positions or offsets, returned or specified, in the LOCATE function are always character offsets and may be different from the byte offset for multibyte data.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

Chapter 4, "Function Support" in *Large Objects Management in Sybase IQ*

LOG function [Numeric]

Function

Returns the natural logarithm of a number.

LN is an alias of LOG.

Syntax

LOG (*numeric-expression*)

Parameters

numeric-expression The number.

Example

The following statement returns the value 3.912023:

```
SELECT LOG( 50 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

"LOG10 function [Numeric]" on page 200

LOG10 function [Numeric]

Function

Returns the base 10 logarithm of a number.

Syntax

LOG10 (*numeric-expression*)

Parameters

numeric-expression The number.

Example

The following statement returns the value 1.698970.

```
SELECT LOG10( 50 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

"LOG function [Numeric]" on page 199

LOWER function [String]

Function

Converts all characters in a string to lowercase.

Syntax

LOWER (*string-expression*)

Parameters

string-expression The string to be converted.

Note The result data type of a LOWER function is a LONG VARCHAR. If you use LOWER in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set LOWER to the correct data type and size.

See "REPLACE function [String]" for more information.

Example

The following statement returns the value "lower case":

```
SELECT LOWER( 'LOWER CasE' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** SQL92 compatible.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

“LCASE function [String]” on page 194

“UCASE function [String]” on page 329

“UPPER function [String]” on page 329

LTRIM function [String]

Function

Removes leading blanks from a string.

Syntax

LTRIM (*string-expression*)

Parameters

string-expression The string to be trimmed.

Note The result data type of an LTRIM function is a LONG VARCHAR. If you use LTRIM in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set LTRIM to the correct data type and size.

See “REPLACE function [String]” on page 242 for more information.

Example

The following statement returns the value “Test Message” with all leading blanks removed:

```
SELECT LTRIM( '     Test Message' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

“RTRIM function [String]” on page 247

“TRIM function [String]” on page 268

MAX function [Aggregate]

Function

Returns the maximum *expression* value found in each group of rows.

Syntax

MAX (*expression*
| **DISTINCT** *column-name*)

Parameters

expression The expression for which the maximum value is to be calculated. This is commonly a column name.

DISTINCT column-name Returns the same as MAX (*expression*), and is included for completeness.

Example

The following statement returns the value 138948.000, representing the maximum salary in the Employees table:

```
SELECT MAX ( Salary )
FROM Employees
```

Usage

Rows where *expression* is NULL are ignored. Returns NULL for a group containing no rows.

Standards and compatibility

- **SQL92** SQL92 compatible.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

"MIN function [Aggregate]" on page 203

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

MEDIAN function [Aggregate]

Function

Returns the median of an expression.

Syntax 1

MEDIAN([ALL | DISTINCT] *expression*)

Syntax 2

MEDIAN([ALL | DISTINCT] *expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

expression A numeric expression for which a median value is to be computed.

Usage

The median is the number separating the higher half of a sample, a population, or a probability distribution, from the lower half.

The data type of the returned value is the same as that of the input value. NULLs are ignored in the calculation of the median value. You can use the optional keyword DISTINCT to eliminate duplicate values before the aggregate function is applied. ALL, which performs the operation on all rows, is the default.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see "Analytical functions" on page 104.

Note The *window-spec* cannot contain a ROW, RANGE or ORDER BY specification; *window-spec* can only specify a PARTITION clause. DISTINCT is not supported if a WINDOW clause is used.

Example

The following query returns the median salary for each department in Florida:

```
SELECT DepartmentID, Surname, Salary,
MEDIAN(Salary) OVER (PARTITION BY DepartmentID)
"Median"
FROM Employees
WHERE State IN ('FL')
```

The returned result is:

Table 4-24: MEDIAN result set

DepartmentID	Surname	Salary	Median
100	Lull	87900.000	73870.000
100	Gowda	59840.000	73870.000
200	Sterling	64900.000	76200.000
200	Kelly	87500.000	76200.000
300	Litton	58930.000	58930.000
400	Francis	53870.000	3870.000
400	Charlton	28300.000	53870.000
400	Evans	68940.000	53870.000

Standards and compatibility

- **SQL2008** Vendor extension

MIN function [Aggregate]

Function

Returns the minimum expression value found in each group of rows.

Syntax

MIN (*expression*
| **DISTINCT** *column-name*)

Parameters

expression The expression for which the minimum value is to be calculated. This is commonly a column name.

DISTINCT column-name Returns the same as MIN (*expression*), and is included for completeness.

Example

The following statement returns the value 24903.000, representing the minimum salary in the Employees table:

```
SELECT MIN ( Salary )
FROM Employees
```

Usage

Rows where *expression* is NULL are ignored. Returns NULL for a group containing no rows.

Standards and compatibility

- **SQL92** SQL92 compatible.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

"MAX function [Aggregate]" on page 201

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

MINUTE function [Date and time]

Function

Returns a number from 0 to 59 corresponding to the minute component of the specified date/time value.

Syntax

MINUTE (*datetime-expression*)

Parameters

datetime-expression The date/time value.

Example

The following statement returns the value 22:

```
SELECT MINUTE( '1998-07-13 12:22:34' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

MINUTES function [Date and time]

Function

Returns the number of minutes since an arbitrary date and time, the number of whole minutes between two specified times, or adds the specified integer-expression number of minutes to a time.

Syntax

MINUTES (*datetime-expression*
| *datetime-expression*, *datetime-expression*
| *datetime-expression*, *integer-expression*)

Parameters

datetime-expression A date and time.

integer-expression The number of minutes to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of minutes are subtracted from the date/time. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a datetime data type.

For information on casting data types, see "CAST function [Data type conversion]" on page 133.

Examples

The following statement returns the value 1051125487:

```
SELECT MINUTES( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 240, to signify the difference between the two times:

```
SELECT MINUTES( '1999-07-13 06:07:12',
   '1999-07-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-05-12 21:10:07.000:

```
SELECT MINUTES( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5) FROM iq_dummy
```

Usage

The second syntax returns the number of whole minutes from the first date/time to the second date/time. The number might be negative.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

MOD function [Numeric]

Function

Returns the remainder when one whole number is divided by another.

Syntax

MOD (*dividend*, *divisor*)

Parameters

dividend The dividend, or numerator of the division.

divisor The divisor, or denominator of the division.

Example

The following statement returns the value 2:

```
SELECT MOD( 5, 3 ) FROM iq_dummy
```

Usage

Division involving a negative *dividend* gives a negative or zero result. The sign of the *divisor* has no effect.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported in Adaptive Server Enterprise. The % operator is used as a modulo operator in Adaptive Server Enterprise

See also

"REMAINDER function [Numeric]" on page 241

MONTH function [Date and time]

Function

Returns a number from 1 to 12 corresponding to the month of the given date.

Syntax

MONTH (*date-expression*)

Parameters

date-expression A date/time value.

Example

The following statement returns the value 7:

```
SELECT MONTH( '1998-07-13' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

MONTHNAME function [Date and time]

Function

Returns the name of the month from the specified date expression.

Syntax

MONTHNAME (*date-expression*)

Parameters

date-expression The datetime value.

Example

The following statement returns the value September, when the DATE_ORDER option is set to the default value of *ymd*.

```
SELECT MONTHNAME( '1998-09-05' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

DATE_ORDER option in *Reference: Statements and Options*

MONTHS function [Date and time]

Function

Returns the number of months since an arbitrary starting date/time or the number of months between two specified date/times, or adds the specified integer-expression number of months to a date/time.

Syntax

MONTHS (*date-expression*
| *date-expression, datetime-expression*
| *date-expression, integer-expression*)

Parameters

date-expression A date and time.

integer-expression The number of months to be added to the *date-expression.* If *integer-expression* is negative, the appropriate number of months are subtracted from the date/time value. If you supply an integer expression, the *date-expression* must be explicitly cast as a datetime data type.

For information on casting data types, see the section "CAST function [Data type conversion]" on page 133.

Examples

The following statement returns the value 23982:

```
SELECT MONTHS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 2, to signify the difference between the two dates:

```
SELECT MONTHS( '1999-07-13 06:07:12',
   '1999-09-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-10-12 21:05:07.000:

```
SELECT MONTHS( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5) FROM iq_dummy
```

Usage

The first syntax returns the number of months since an arbitrary starting date. This number is often useful for determining whether two date/time expressions are in the same month in the same year.

```
MONTHS( invoice_sent ) = MONTHS( payment_received )
```

Note that comparing the MONTH function would incorrectly include a payment made 12 months after the invoice was sent.

The second syntax returns the number of months from the first date to the second date. The number might be negative. It is calculated from the number of the first days of the month between the two dates. Hours, minutes and seconds are ignored.

The third syntax adds *integer-expression* months to the given date. If the new date is past the end of the month (such as MONTHS ('1992-01-31', 1)) the result is set to the last day of the month. If *integer-expression* is negative, the appropriate number of months are subtracted from the date. Hours, minutes and seconds are ignored.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

NEWID function [Miscellaneous]

Function

Generates a UUID (Universally Unique Identifier) value. The returned UUID value is a binary. A UUID is the same as a GUID (Globally Unique Identifier).

Syntax

NEWID ( )

Parameters

There are no parameters associated with NEWID().

Example

The following statement creates the table t1 and then updates the table, setting the value of the column uid_col to a unique identifier generated by the NEWID function, if the current value of the column is NULL.

```
CREATE TABLE t1 (uid_col int);
UPDATE t1
    SET uid_col = NEWID()
      WHERE uid_col IS NULL
```

If you execute the following statement,

```
SELECT NEWID()
```

the unique identifier is returned as a BINARY(16). For example, the value might be 0xd3749fe09cf446e399913bc6434f1f08. You can convert this string into a readable format using the UUIDTOSTR() function.

Usage

The NEWID() function generates a unique identifier value.

UUIDs can be used to uniquely identify objects in a database. The values are generated such that a value produced on one computer does not match that produced on another, hence they can also be used as keys in replication and synchronization environments.

The NEWID function is supported only in the following positions:

- SELECT list of a top level query block
- SET clause of an UPDATE statement
- VALUES clause of INSERT...VALUES

You can use a value generated by the NEWID function as a column default value in a Sybase IQ table.

Standards and compatibility

- **SQL92** Vendor extension
- **SQL99** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise.

See also

"STRTOUUID function [String]" on page 263

"UUIDTOSTR function [String]" on page 331

“Binary data types” on page 77

“Character data types” on page 69

NEXT_CONNECTION function [System]

Function

Returns the next connection number, or the first connection if the parameter is NULL.

Syntax

NEXT_CONNECTION ({**NULL** | *connection-id* })

Note CIS functional compensation performance considerations apply. See “Conditions that cause processing by SQL Anywhere” in *Chapter 3, “Optimizing Queries and Deletions,”* in the *Performance and Tuning Guide*.

Parameters

connection-id An integer, usually returned from a previous call to NEXT_CONNECTION. If *connection-id* is NULL, NEXT_CONNECTION returns the first connection ID.

Examples

The following statement returns an identifier for the first connection. The identifier is an integer value like 569851433.

```
SELECT NEXT_CONNECTION( NULL ) FROM iq_dummy
```

The following statement returns a value like 1661140050:

```
SELECT NEXT_CONNECTION( 569851433 ) FROM iq_dummy
```

Usage

You can use NEXT_CONNECTION to enumerate the connections to a database. To get the first connection, pass NULL; to get each subsequent connection, pass the previous return value. The function returns NULL when there are no more connections.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

NEXT_DATABASE function [System]

Function

Returns the next database ID number, or the first database if the parameter is NULL.

Syntax

NEXT_DATABASE ({ **NULL** | *database-id* })

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

database-id An integer that specifies the ID number of the database.

Examples

The following statement returns the value 0, the first database value:

```
SELECT NEXT_DATABASE( NULL ) FROM iq_dummy
```

The following statement returns NULL, indicating that there are no more databases on the server:

```
SELECT NEXT_DATABASE( 0 ) FROM iq_dummy
```

Usage

You can use NEXT_DATABASE to enumerate the databases running on a database server. To get the first database, pass NULL; to get each subsequent database, pass the previous return value. The function returns NULL when there are no more databases.

Standards and compatibility

- **SQL92** Transact-SQL extension.
- **Sybase** Not supported by Adaptive Server Enterprise.

See also

"OBJECT_ID function [System]" on page 215

NEXT_HTTP_HEADER function [HTTP]

Function

Gets the next HTTP header name.

Syntax

NEXT_HTTP_HEADER (*header-name*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

header-name The name of the previous header. If header-name is null, this function returns the name of the first HTTP header.

Usage

This function iterates over the HTTP headers included within a request. Calling it with NULL causes it to return the name of the first header. Subsequent headers are retrieved by passing the function the name of the previous header. This function returns NULL when called with the name of the last header.

Calling this function repeatedly returns all the header fields exactly once, but not necessarily in the order in which they appear in the HTTP request.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

NEXT_HTTP_VARIABLE function [HTTP]

Function

Get the next HTTP variable name.

Syntax

NEXT_HTTP_VARIABLE (*var-name*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

var-name The name of the previous variable. If var-name is null, this function returns the name of the first HTTP variable.

Usage

This function iterates over the HTTP variables included within a request. Calling it with NULL causes it to return the name of the first variable. Subsequent variables are retrieved by passing the function the name of the previous variable. This function returns NULL when called with the name of the final variable.

Calling this function repeatedly returns all the variables exactly once, but not necessarily in the order in which they appear in the HTTP request. The variables *url* or *url1*, *url2*, …, *url10* are included if URL PATH is set to ON or ELEMENTS, respectively.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"HTML_DECODE function [HTTP]" on page 181

"HTTP_VARIABLE function [HTTP]" on page 187

"NEXT_HTTP_HEADER function [HTTP]" on page 210

NOW function [Date and time]

Function

Returns the current date and time. This is the historical syntax for CURRENT TIMESTAMP.

Syntax **NOW** (*****)

Example The following statement returns the current date and time.

```
SELECT NOW(*) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise.

NTILE function [Analytical]

Function Distributes query results into a specified number of “buckets” and assigns the bucket number to each row in the bucket.

Syntax

NTILE (*expression1*)
OVER (**ORDER BY** *expression2* [**ASC** | **DESC**])

Parameters **expression1** A constant integer from 1 to 32767, which specifies the number of buckets.

expression2 A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Example The following example uses the NTILE function to determine the sale status of car dealers. The dealers are divided into four groups based on the number of cars each dealer sold. The dealers with ntile = 1 are in the top 25% for car sales.

```
SELECT dealer_name, sales,
NTILE(4) OVER ( ORDER BY sales DESC )
FROM carSales;


dealer_name         sales       ntile
Boston              1000        1
Worcester           950         1
Providence          950         1
SF                  940         1
Lowell              900         2
Seattle             900         2
Natick              870         2
New Haven           850         2
Portland            800         3
Houston             780         3
Hartford            780         3
Dublin              750         3
Austin              650         4
Dallas              640         4
```

```
Dover                  600          4
```

To find the top 10% of car dealers by sales, you specify NTILE(10) in the example SELECT statement. Similarly, to find the top 50% of car dealers by sales, specify NTILE(2).

Usage

NTILE is a rank analytical function that distributes query results into a specified number of buckets and assigns the bucket number to each row in the bucket. You can divide a result set into one-hundredths (percentiles), tenths (deciles), fourths (quartiles), or other numbers of groupings.

NTILE requires an OVER (ORDER BY) clause. The ORDER BY clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. Note that this ORDER BY clause is used only within the OVER clause and is *not* an ORDER BY for the SELECT. No aggregation functions in the rank query are allowed to specify DISTINCT.

The OVER clause indicates that the function operates on a query result set. The result set is the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses have all been evaluated. The OVER clause defines the data set of the rows to include in the computation of the rank analytical function.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

NTILE is allowed only in the select list of a SELECT or INSERT statement or in the ORDER BY clause of the SELECT statement. NTILE can be in a view or a union. The NTILE function cannot be used in a subquery, a HAVING clause, or in the select list of an UPDATE or DELETE statement. Only one NTILE function is allowed per query.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

"Analytical functions" on page 104

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

NULLIF function [Miscellaneous]

Function

Provides an abbreviated CASE expression by comparing expressions.

Syntax

NULLIF (*expression1*, *expression2*)

Parameters

expression1 An expression to be compared.

expression2 An expression to be compared.

Examples

The following statement returns a:

```
SELECT NULLIF( 'a', 'b' ) FROM iq_dummy
```

The following statement returns NULL:

```
SELECT NULLIF( 'a', 'a' ) FROM iq_dummy
```

Usage

NULLIF compares the values of the two expressions.

If the first expression equals the second expression, NULLIF returns NULL.

If the first expression does not equal the second expression, or if the second expression is NULL, NULLIF returns the first expression.

The NULLIF function provides a short way to write some CASE expressions. NULLIF is equivalent to:

CASE WHEN *expression1* = *expression2* **THEN NULL**
ELSE *expression1* **END**

Standards and compatibility

- **SQL92** Transact-SQL extension.
- **Sybase** Not supported by Adaptive Server Enterprise.

See also

"CASE expressions" on page 29

NUMBER function [Miscellaneous]

Function

Generates numbers starting at 1 for each successive row in the results of the query.

Syntax

NUMBER (*****)

Example

The following statement returns this numbered list:

number(*)
1
2
3
4
5

```
SELECT NUMBER( * )
FROM Departments
WHERE DepartmentID > 10
```

Usage

Use the NUMBER function only in a select list or a SET clause of an UPDATE statement. For example, the following statement updates each row of the seq_id column with a number 1 greater than the previous row. The number is applied in the order specified by the ORDER BY clause.

```
update empl
set seq_id = number(*)
order by empl_id
```

In an UPDATE statement, if the NUMBER(*) function is used in the SET clause and the FROM clause specifies a one-to-many join, NUMBER(*) generates unique numbers that increase, but may not increment sequentially due to row elimination.

NUMBER can also be used to generate primary keys when using the INSERT from SELECT statement, although using IDENTITY/AUTOINCREMENT is a preferred mechanism for generating sequential primary keys.

Note A syntax error is generated if you use NUMBER in a DELETE statement, WHERE clause, HAVING clause, ORDER BY clause, subquery, query involving aggregation, any constraint, GROUP BY, DISTINCT, a query containing UNION ALL, or a derived table.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise.

OBJECT_ID function [System]

Function

Returns the object ID.

Syntax

OBJECT_ID (*object-name*)

Parameters

object-name The name of the object.

Examples

The following statement returns the object ID 100209 of the *Customers* table:

```
SELECT OBJECT_ID ('CUSTOMERS') FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ.

See also

“COL_NAME function [System]” on page 138

“DB_ID function [System]” on page 161

“OBJECT_NAME function [System]” on page 216

OBJECT_NAME function [System]

Function: Returns the object name.

Syntax:

OBJECT_NAME (*object-id* [, *database-id*])

Parameters:

object-id The object ID.

database-id The database ID.

Examples: The following statement returns the name “customer:”

```
SELECT OBJECT_NAME ( 100209 ) FROM iq_dummy
```

Standards and compatibility:

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ.

See also:

“COL_NAME function [System]” on page 138

“DB_NAME function [System]” on page 162

“OBJECT_ID function [System]” on page 215

OCTET_LENGTH function [String]

Function: Returns an unsigned 64-bit value containing the byte length of the column parameter.

Syntax:

OCTET_LENGTH(*column-name*)

Parameters:

column-name The name of a column.

Usage: The return value of a NULL argument is NULL.

The OCTET_LENGTH function supports all Sybase IQ data types.

Standards and compatibility: **Sybase** Not supported by SQL Anywhere or Adaptive Server Enterprise.

See also:

“BIT_LENGTH function [String]” on page 132

PATINDEX function [String]

Function

Returns the starting position of the first occurrence of a specified pattern.

Syntax

PATINDEX (**'%***pattern***%'**, *string-expression*)

Parameters

pattern The pattern for which you are searching. This string is limited to 126 bytes for patterns with wildcards. If the leading percent wildcard is omitted, PATINDEX returns one (1) if the pattern occurs at the beginning of the string, and zero if not. If *pattern* starts with a percent wildcard, then the two leading percent wildcards are treated as one.

The pattern uses the same wildcards as the LIKE comparison. Table 4-25 lists the pattern wildcards.

Table 4-25: PATINDEX pattern wildcards

Wildcard	Matches
_ (underscore)	Any one character
% (percent)	Any string of zero or more characters
[]	Any single character in the specified range or set
[^]	Any single character not in the specified range or set

Patterns without wildcards (percent % or underscore _) can be up to 255 bytes in length.

string-expression The string to be searched for the pattern.

Examples

The following statement returns the value 2:

```
SELECT PATINDEX( '%hoco%', 'chocolate' ) FROM iq_dummy
```

The following statement returns the value 11:

```
SELECT PATINDEX ('%4_5_', '0a1A 2a3A 4a5A') FROM
iq_dummy
```

Usage

PATINDEX returns the starting position of the first occurrence of the pattern. If the string being searched contains more than one instance of the string pattern, PATINDEX returns only the position of the first instance.

If the pattern is not found, PATINDEX returns zero (0).

Searching for a pattern longer than 126 bytes returns NULL.

Searching for a zero-length string returns 1.

If any of the arguments is NULL, the result is zero (0).

All the positions or offsets, returned or specified, in the PATINDEX function are always character offsets and may be different from the byte offset for multibyte data.

PATINDEX returns a 32 bit unsigned integer position for CHAR and VARCHAR columns.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

“LIKE conditions” on page 40

“LOCATE function [String]” on page 198

Chapter 4, “Function Support” in *Large Objects Management in Sybase IQ*

PERCENT_RANK function [Analytical]

Function

Computes the (fractional) position of one row returned from a query with respect to the other rows returned by the query, as defined by the ORDER BY clause. Returns a decimal value between 0 and 1.

Syntax

PERCENT_RANK () **OVER** (**ORDER BY** *expression* [**ASC** | **DESC**])

Parameters

expression A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Example

The following statement illustrates the use of the PERCENT_RANK function:

```
SELECT s_suppkey, SUM(s_acctBal) AS sum_acctBal,
PERCENT_RANK() OVER ( ORDER BY SUM(s_acctBal) DESC )
AS percent_rank_all FROM supplier GROUP BY s_suppkey;

s_suppkey        sum_acctBal       percent_rank_all
supplier#011     200000            0
supplier#002     200000            0
supplier#013     123000            0.3333
supplier#004     110000            0.5
supplier#035     110000            0.5
supplier#006     50000             0.8333
supplier#021     10000             1
```

Usage

PERCENT_RANK is a rank analytical function. The percent rank of a row R is defined as the rank of a row in the groups specified in the OVER clause minus one divided by the number of total rows in the groups specified in the OVER clause minus one. PERCENT_RANK returns a value between 0 and 1. The first row has a percent rank of zero.

The PERCENT_RANK of a row is calculated as

```
(Rx - 1) / (NtotalRow - 1)
```

where *Rx* is the rank position of a row in the group and *NtotalRow* is the total number of rows in the group specified by the OVER clause.

PERCENT_RANK requires an OVER (ORDER BY) clause. The ORDER BY clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the OVER clause and is *not* an ORDER BY for the SELECT. No aggregation functions in the rank query are allowed to specify DISTINCT.

The OVER clause indicates that the function operates on a query result set. The result set is the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses have all been evaluated. The OVER clause defines the data set of the rows to include in the computation of the rank analytical function.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

PERCENT_RANK is allowed only in the select list of a SELECT or INSERT statement or in the ORDER BY clause of the SELECT statement. PERCENT_RANK can be in a view or a union. The PERCENT_RANK function cannot be used in a subquery, a HAVING clause, or in the select list of an UPDATE or DELETE statement. Only one rank analytical function is allowed per query.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

"Analytical functions" on page 104

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

PERCENTILE_CONT function [Analytical]

Function

Given a percentile, returns the value that corresponds to that percentile. Assumes a continuous distribution data model.

Note If you are simply trying to compute a percentile, use the NTILE function instead, with a value of 100.

Syntax

PERCENTILE_CONT (*expression1*)
WITHIN GROUP (**ORDER BY** *expression2* [**ASC** | **DESC**])

Parameters

expression1 A constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, a "wrong argument for percentile" error is returned. If the argument value is less than 0 or greater than 1, a "data value out of range" error is returned.

expression2 A sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

Example

The following example uses the PERCENTILE_CONT function to determine the 10th percentile value for car sales in a region.

The following data set is used in the example:

```
sales       region        dealer_name
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
700         Northeast     Lowell
540         Northeast     Natick
500         Northeast     New Haven
450         Northeast     Hartford
800         Northwest     SF
600         Northwest     Seattle
500         Northwest     Portland
400         Northwest     Dublin
500         South         Houston
400         South         Austin
300         South         Dallas
200         South         Dover
```

The following SELECT statement contains the PERCENTILE_CONT function:

```
SELECT region, PERCENTILE_CONT(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
```

```
FROM carSales GROUP BY region;
```

The result of the SELECT statement lists the 10th percentile value for car sales in a region:

```
region          percentile_cont
Northeast       840
Northwest       740
South           470
```

Usage

The inverse distribution analytical functions return a k-th percentile value, which can be used to help establish a threshold acceptance value for a set of data. The function PERCENTILE_CONT takes a percentile value as the function argument, and operates on a group of data specified in the WITHIN GROUP clause, or operates on the entire data set. The function returns one value per group. If the GROUP BY column from the query is not present, the result is a single row. The data type of the results is the same as the data type of its ORDER BY item specified in the WITHIN GROUP clause. The data type of the ORDER BY expression for PERCENTILE_CONT must be numeric.

PERCENTILE_CONT requires a WITHIN GROUP (ORDER BY) clause.

The ORDER BY clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. For the PERCENTILE_CONT function, the data type of this expression must be numeric. This ORDER BY clause is used only within the WITHIN GROUP clause and is *not* an ORDER BY for the SELECT.

The WITHIN GROUP clause distributes the query result into an ordered data set from which the function calculates a result. The WITHIN GROUP clause must contain a single sort item. If the WITHIN GROUP clause contains more or less than one sort item, an error is reported.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

The PERCENTILE_CONT function is allowed in a subquery, a HAVING clause, a view, or a union. PERCENTILE_CONT can be used anywhere the simple nonanalytical aggregate functions are used. The PERCENTILE_CONT function ignores the NULL value in the data set.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

“Analytical functions” on page 104

“NTILE function [Analytical]” on page 212

"PERCENTILE_DISC function [Analytical]" on page 222

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

PERCENTILE_DISC function [Analytical]

Function

Given a percentile, returns the value that corresponds to that percentile. Assumes a discrete distribution data model.

Note If you are simply trying to compute a percentile, use the NTILE function instead, with a value of 100.

Syntax

PERCENTILE_DISC (*expression1*)
WITHIN GROUP (**ORDER BY** *expression2* [**ASC** | **DESC**])

Parameters

expression1 A constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, then a "wrong argument for percentile" error is returned. If the argument value is less than 0 or greater than 1, then a "data value out of range" error is returned.

expression2 A sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

Example

The following example uses the PERCENTILE_DISC function to determine the 10th percentile value for car sales in a region.

The following data set is used in the example:

```
sales       region        dealer_name
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
700         Northeast     Lowell
540         Northeast     Natick
500         Northeast     New Haven
450         Northeast     Hartford
800         Northwest     SF
600         Northwest     Seattle
500         Northwest     Portland
400         Northwest     Dublin
500         South         Houston
400         South         Austin
300         South         Dallas
```

```
200          South          Dover
```

The following SELECT statement contains the PERCENTILE_DISC function:

```
SELECT region, PERCENTILE_DISC(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

The result of the SELECT statement lists the 10th percentile value for car sales in a region:

```
region           percentile_cont
Northeast        900
Northwest        800
South            500
```

Usage

The inverse distribution analytical functions return a k-th percentile value, which can be used to help establish a threshold acceptance value for a set of data. The function PERCENTILE_DISC takes a percentile value as the function argument and operates on a group of data specified in the WITHIN GROUP clause or operates on the entire data set. The function returns one value per group. If the GROUP BY column from the query is not present, the result is a single row. The data type of the results is the same as the data type of its ORDER BY item specified in the WITHIN GROUP clause. PERCENTILE_DISC supports all data types that can be sorted in Sybase IQ.

PERCENTILE_DISC requires a WITHIN GROUP (ORDER BY) clause.

The ORDER BY clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the WITHIN GROUP clause and is *not* an ORDER BY for the SELECT.

The WITHIN GROUP clause distributes the query result into an ordered data set from which the function calculates a result. The WITHIN GROUP clause must contain a single sort item. If the WITHIN GROUP clause contains more or less than one sort item, an error is reported.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

The PERCENTILE_DISC function is allowed in a subquery, a HAVING clause, a view, or a union. PERCENTILE_DISC can be used anywhere the simple nonanalytical aggregate functions are used. The PERCENTILE_DISC function ignores the NULL value in the data set.

Standards and compatibility

- **SQL92** Vendor extension

- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

"Analytical functions" on page 104

"NTILE function [Analytical]" on page 212

"PERCENTILE_CONT function [Analytical]" on page 220

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

PI function [Numeric]

Function

Returns the numeric value PI.

Syntax

PI (*)

Example

The following statement returns the value 3.141592653....

```
SELECT PI( * ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** The PI() function is supported in Adaptive Server Enterprise, but PI(*) is not.

POWER function [Numeric]

Function

Calculates one number raised to the power of another.

Syntax

POWER (*numeric-expression1*, *numeric-expression2*)

Parameters

numeric-expression1 The base.

numeric-expression2 The exponent.

Example

The following statement returns the value 64:

```
SELECT Power( 2, 6 ) FROM iq_dummy
```

Usage

Raises *numeric-expression1* to the power *numeric-expresson2*.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise.

PROPERTY function [System]

Function

Returns the value of the specified server-level property as a string.

Syntax

PROPERTY ({ *property-id* | *property-name* })

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

property-id An integer that is the property-number of the server-level property. This number can be determined from the PROPERTY_NUMBER function. The *property-id* is commonly used when looping through a set of properties.

property-name A string giving the name of the property. See "Properties available for the server" on page 121 for a list of server property names.

Example

The following statement returns the name of the current database server:

```
SELECT PROPERTY( 'Name' ) FROM iq_dummy
```

Usage

Each property has both a number and a name, but the number is subject to change between versions, and should not be used as a reliable identifier for a given property.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise.

See also

"Properties available for the server" on page 121

PROPERTY_DESCRIPTION function [System]

Function

Returns a description of a property.

Syntax

PROPERTY_DESCRIPTION ({ *property-id* | *property-name* })

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

property-id An integer that is the property number of the property. This number can be determined from the PROPERTY_NUMBER function. The *property-id* is commonly used when looping through a set of properties.

property-name A string giving the name of the property. For property names, see the lists in "Connection properties" on page 121, "Properties available for the server" on page 121, and "Properties available for each database" on page 122.

Example

The following statement returns the description "Number of index insertions:"

```
SELECT PROPERTY_DESCRIPTION( 'IndAdd' ) FROM iq_dummy
```

Usage

Each property has both a number and a name, but the number is subject to change between releases, and should not be used as a reliable identifier for a given property.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise.

See also

"Connection properties" on page 121

"Properties available for the server" on page 121

"Properties available for each database" on page 122

PROPERTY_NAME function [System]

Function

Returns the name of the property with the supplied property number.

Syntax

PROPERTY_NAME (*property-id*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters

property-id The property number of the property.

Example

The following statement returns the property associated with property number 126. The actual property to which this refers changes from version to version.

```
SELECT PROPERTY_NAME( 126 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise.

See also

"Connection properties" on page 121

"Properties available for the server" on page 121

"Properties available for each database" on page 122

PROPERTY_NUMBER function [System]

Function
: Returns the property number of the property with the supplied property name.

Syntax
: **PROPERTY_NUMBER** (*property-name*)

Note CIS functional compensation performance considerations apply. See "Conditions that cause processing by SQL Anywhere" in *Chapter 3, "Optimizing Queries and Deletions,"* in the *Performance and Tuning Guide*.

Parameters
: **property-name** A property name. For property names, see the lists in "Connection properties" on page 121, "Properties available for the server" on page 121, and "Properties available for each database" on page 122.

Example
: The following statement returns an integer value. The actual value changes from version to version.

```
SELECT PROPERTY_NUMBER( 'PAGESIZE' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"Connection properties" on page 121

"Properties available for the server" on page 121

"Properties available for each database" on page 122

QUARTER function [Date and time]

Function
: Returns a number indicating the quarter of the year from the supplied date expression.

Syntax
: **QUARTER**(*date-expression*)

Parameters
: **date-expression** A date.

Example
: With the DATE_ORDER option set to the default of *ymd*, the following statement returns the value 2:

```
SELECT QUARTER ( '1987/05/02' ) FROM iq_dummy
```

Usage
: Table 4-26 lists the dates in the quarters of the year.

Table 4-26: Values of quarter of the year

Quarter	Period (inclusive)
1	January 1 to March 31
2	April 1 to June 30
3	July 1 to September 30
4	October 1 to December 31

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

DATE_ORDER option in *Reference: Statements and Options*

RADIANS function [Numeric]

Function

Converts a number from degrees to radians.

Syntax

RADIANS (*numeric-expression*)

Parameters

numeric-expression A number, in degrees. This angle is converted to radians.

Example

The following statement returns a value of approximately 0.5236:

```
SELECT RADIANS( 30 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

RAND function [Numeric]

Function

Returns a DOUBLE precision, random number x, where 0 <= x <1, with an optional seed.

Syntax

RAND ([*integer-expression*])

Parameters

integer-expression The optional seed used to create a random number. This argument allows you to create repeatable random number sequences.

If RAND is called with a FROM clause and an argument in a query containing only tables in IQ stores, the function returns an arbitrary but repeatable value.

When no argument is called, RAND is a non-deterministic function. Successive calls to RAND might return different values. The query optimizer does not cache the results of the RAND function

---

Note The values returned by RAND vary depending on whether you use a FROM clause or not and whether the referenced table was created in SYSTEM or in an IQ store.

Examples

The following statement returns a 5% sampling of a table:

```
SELECT AVG(table1.number_of_cars),
AVG(table1.number_of_tvs)FROM table1 WHERE
RAND(ROWID(table1)) < .05 and table1.income < 50000;
```

The following statement returns a value of approximately 941392926249216914:

```
SELECT RAND( 4 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

RANK function [Analytical]

Function

Ranks items in a group.

Syntax

RANK () **OVER** (**ORDER BY** *expression* [**ASC** | **DESC**])

Parameters

expression A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Example

The following statement illustrates the use of the RANK function:

```
SELECT s_suppkey, SUM(s_acctBal) AS sum_acctBal,
RANK() OVER ( ORDER BY SUM(s_acctBal) DESC )
AS rank_all FROM supplier GROUP BY s_suppkey;

s_suppkey        sum_acctBal       rank_all
supplier#011     200000            1
supplier#002     200000            1
supplier#013     123000            3
supplier#004     110000            4
supplier#035     110000            4
supplier#006     50000             6
supplier#021     10000             7
```

Usage

RANK is a rank analytical function. The rank of row R is defined as the number of rows that precede R and are not peers of R. If two or more rows are not distinct within the groups specified in the OVER clause or distinct over the entire result set, then there are one or more gaps in the sequential rank numbering. The difference between RANK and DENSE_RANK is that DENSE_RANK leaves no gap in the ranking sequence when there is a tie. RANK leaves a gap when there is a tie.

RANK requires an OVER (ORDER BY) clause. The ORDER BY clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the OVER clause and is *not* an ORDER BY for the SELECT. No aggregation functions in the rank query are allowed to specify DISTINCT.

The OVER clause indicates that the function operates on a query result set. The result set is the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses have all been evaluated. The OVER clause defines the data set of the rows to include in the computation of the rank analytical function.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

RANK is allowed only in the select list of a SELECT or INSERT statement or in the ORDER BY clause of the SELECT statement. RANK can be in a view or a union. The RANK function cannot be used in a subquery, a HAVING clause, or in the select list of an UPDATE or DELETE statement. Only one rank analytical function is allowed per query.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere

See also

"Analytical functions" on page 104

"DENSE_RANK function [Analytical]" on page 163

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

REGR_AVGX function [Aggregate]

Function

Computes the average of the independent variable of the regression line.

Syntax 1

REGR_AVGX (*dependent-expression*, *independent-expression*)

Syntax 2

REGR_AVGX (*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then REGR_AVGX returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where x represents the independent-expression:

AVG (x)

See "Mathematical formulas for the aggregate functions" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions*.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see "Analytical functions" on page 104.

Example

The following example calculates the average of the dependent variable, employee age:

```
SELECT REGR_AVGX( Salary, ( YEAR( NOW() ) - YEAR( 
BirthDate ) ) )

FROM Employees;
```

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

REGR_AVGY function [Aggregate]

Function

Computes the average of the dependent variable of the regression line.

Syntax 1

REGR_AVGY(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_AVGY(*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then REGR_AVGY returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where y represents the dependent-expression:

AVG(y)

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions*.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example calculates the average of the independent variable, employee salary. This function returns the value 49988.6232:

```
SELECT REGR_AVGY( Salary, ( YEAR( NOW( )) - YEAR(
BirthDate ) ) )
```

```
FROM Employees;
```

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

REGR_COUNT function [Aggregate]

Function

Returns an integer that represents the number of non-NULL number pairs used to fit the regression line.

Syntax 1

REGR_COUNT(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_COUNT(*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage

This function returns an UNSIGNED BIGINT as the result.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example returns a value that indicates the number of non-NULL pairs that were used to fit the regression line. This function returns the value 75:

```
SELECT REGR_COUNT( Salary, ( YEAR( NOW() ) - YEAR(
BirthDate ) ) )
FROM Employees;
```

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQ.
- **Sybase** Compatible with SQL Anywhere

REGR_INTERCEPT function [Aggregate]

Function

Computes the y-intercept of the linear regression line that best fits the dependent and independent variables.

Syntax 1

REGR_INTERCEPT(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_INTERCEPT(*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, REGR_INTERCEPT returns NULL.

The function is applied to the set of (*dependent-expression* and *independent-expression*) pairs after eliminating all pairs for which either *dependent-expression* or *independent-expression* is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the *dependent-expression* and x represents the *independent-expression*:

AVG(y) - REGR_SLOPE(y, x) * AVG(x)

See "Mathematical formulas for the aggregate functions" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions*.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see "Analytical functions" on page 104.

Example

The following example returns the value 1874.5805688517603:

```
SELECT REGR_INTERCEPT( Salary, ( YEAR( NOW() ) - YEAR(
BirthDate ) ) )
```

```
FROM Employees;
```

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

REGR_R2 function [Aggregate]

Function

Computes the coefficient of determination (also referred to as R-squared or the goodness-of-fit statistic) for the regression line.

Syntax 1

REGR_R2(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_R2(*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then REGR_R2 returns NULL.

REGR_R2 is applied to the set of (*dependent-expression* and *independent-expression*) pairs after eliminating all pairs for which either *dependent-expression* or *independent-expression* is NULL. IQ then applies the following algorithm:

- REGR_R2 calculates *VAR_POP(x)* and returns NULL if *VAR_POP(x) = 0*; otherwise, it calculates *VAR_POP(y)* and returns the value 1 if *VAR_POP(y) = 0*.
- If neither *VAR_POP(x)* or *VAR_POP(y)* is zero, the return value is *POWER(CORR(y,x),2)*

where y represents the *dependent-expression* and x represents the *independent-expression*.

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions*.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example returns the value 0.19379959710325653:

```
SELECT REGR_R2( Salary, ( YEAR( NOW() ) - YEAR(
BirthDate ) ) )

FROM Employees;
```

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

REGR_SLOPE function [Aggregate]

Function

Computes the slope of the linear regression line, fitted to non-NULL pairs.

Syntax 1

REGR_SLOPE(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_SLOPE(*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then REGR_SLOPE returns NULL.

REGR_SLOPE is applied to the set of (*dependent-expression* and *independent-expression*) pairs after eliminating all pairs for which either *dependent-expression* or *independent-expression* is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the *dependent-expression* and x represents the *independent-expression*:

COVAR_POP(x, y) / VAR_POP(y)

See "Mathematical formulas for the aggregate functions" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions*.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see "Analytical functions" on page 104.

Example

The following example returns the value 935.3429749445614:

```
SELECT REGR_SLOPE( Salary, ( YEAR( NOW() ) - YEAR(
BirthDate ) ) )

FROM Employees;
```

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

REGR_SXX function [Aggregate]

Function

Computes the slope of the linear regression line, fitted to non-NULL pairs.

Syntax 1

REGR_SXX(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_SXX(*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then REGR_SXX returns NULL.

The function is applied to the set of (*dependent-expression* and *independent-expression*) pairs after eliminating all pairs for which either *dependent-expression* or *independent-expression* is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the *dependent-expression* and x represents the *independent-expression*:

REGR_COUNT(y, x) * VAR_POP(x)

See "Mathematical formulas for the aggregate functions" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions*.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see "Analytical functions" on page 104.

Example

The following example returns the value 5916.4800000000105:

```
SELECT REGR_SXX( Salary, ( YEAR( NOW() ) - YEAR(
BirthDate ) ) )

FROM Employees;
```

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

REGR_SXY function [Aggregate]

Function

Returns the sum of products of the dependent and independent variables. Use REGR_SXY to evaluate the statistical validity of a regression model.

Syntax 1

REGR_SXY(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_SXY(*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then it returns NULL.

The function is applied to the set of (*dependent-expression* and *independent-expression*) pairs after eliminating all pairs for which either *dependent-expression* or *independent-expression* is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the *dependent-expression* and x represents the *independent-expression*:

REGR_COUNT(x, y) * COVAR_POP(x, y)

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions*.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example returns the value 5533938.004400015.

```
SELECT REGR_SXY( Salary, ( YEAR( NOW() ) - YEAR(
BirthDate ) ) )

FROM Employees;
```

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

REGR_SYY function [Aggregate]

Function
Returns values that can evaluate the statistical validity of a regression model.

Syntax 1
REGR_SYY(*dependent-expression*, *independent-expression*)

Syntax 2
REGR_SYY(*dependent-expression*, *independent-expression*)
OVER (*window-spec*)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters
dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage
This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then REGR_SYY returns NULL.

The function is applied to the set of (*dependent-expression* and *independent-expression*) pairs after eliminating all pairs for which either *dependent-expression* or *independent-expression* is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where y represents the *dependent-expression* and x represents the *independent-expression*:

REGR_COUNT(x, y) * VAR_POP(y)

See "Mathematical formulas for the aggregate functions" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions*.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see "Analytical functions" on page 104.

Example
The following example returns the value 26, 708, 672,843.3002:

```
SELECT REGR_SYY( Salary, ( YEAR( NOW() ) - YEAR(
BirthDate ) ) )

FROM Employees;
```

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

REMAINDER function [Numeric]

Function
Returns the remainder when one whole number is divided by another.

Syntax
REMAINDER (*dividend*, *divisor*)

Parameters
dividend The dividend, or numerator of the division.

divisor The divisor, or denominator of the division.

Example
The following statement returns the value 2:

```
SELECT REMAINDER( 5, 3 ) FROM iq_dummy
```

Usage
REMAINDER is the same as the MOD function.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported in Adaptive Server Enterprise. The % (modulo) operator and the division operator can be used to produce a remainder.

See also
"MOD function [Numeric]" on page 205

REPEAT function [String]

Function
Concatenates a string a specified number of times.

Syntax
REPEAT (*string-expression*, *integer-expression*)

Parameters
string-expression The string to be repeated.

integer-expression The number of times the string is to be repeated. If *integer-expression* is negative, an empty string is returned.

Note The result data type of a REPEAT function is a LONG VARCHAR. If you use REPEAT in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set REPEAT to the correct data type and size.

See "REPLACE function [String]" for more information.

Example
The following statement returns the value "repeatrepeatrepeat:"

```
SELECT REPEAT( 'repeat', 3 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported in Adaptive Server Enterprise, but REPLICATE provides the same capabilities

See also

"REPLICATE function [String]" on page 243

REPLACE function [String]

Function

Replaces all occurrences of a substring with another substring.

Syntax

REPLACE (*original-string*, *search-string*, *replace-string*)

Parameters

If any argument is NULL, the function returns NULL.

original-string The string to be searched. This string can be any length.

search-string The string to be searched for and replaced with *replace-string*. This string is limited to 255 bytes. If *search-string* is an empty string, the original string is returned unchanged.

replace-string The replacement string, which replaces *search-string*. This can be any length. If *replace-string* is an empty string, all occurrences of *search-string* are deleted.

If you need to control the width of the resulting column when *replace-string* is wider than *search-string*, use the CAST function. For example,

```
CREATE TABLE aa(a CHAR(5));
INSERT INTO aa VALUES('CCCCC');
COMMIT;
SELECT a, CAST(REPLACE(a,'C','ZZ') AS CHAR(5)) FROM aa;
```

Examples

The following statement returns the value "xx.def.xx.ghi:"

```
SELECT REPLACE( 'abc.def.abc.ghi', 'abc', 'xx' ) FROM
iq_dummy
```

The following statement generates a result set containing ALTER PROCEDURE statements which, when executed, repair stored procedures that reference a table that has been renamed. (To be useful, the table name must be unique.)

```
SELECT REPLACE(
   replace(proc_defn,'OldTableName','NewTableName'),
   'create procedure',
   'alter procedure')
FROM SYS.SYSPROCEDURE
WHERE proc_defn LIKE '%OldTableName%'
```

Use a separator other than the comma for the LIST function:

```
SELECT REPLACE( list( table_id ), ',', '--')
FROM  SYS.ISYSTAB
WHERE table_id <= 5
```

Usage

The result data type of a REPLACE function is a LONG VARCHAR. If you use REPLACE in a SELECT INTO statement, you must have a Large Objects Management option license, or use CAST and set REPLACE to the correct data type and size.

There are two ways to work around this issue:

- Declare a local temporary table, then perform an INSERT:

```
DECLARE local temporary table #mytable
   (name_column char(10)) on commit preserve rows;
INSERT INTO #mytable SELECT REPLACE(name,'0','1')
   FROM dummy_table01;
```

- Use CAST:

```
SELECT CAST(replace(name, '0', '1') AS Char(10))
into #mytable from dummy_table01;
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

"SUBSTRING function [String]" on page 265

REPLICATE function [String]

Function

Concatenates a string a specified number of times.

Syntax

REPLICATE (*string-expression*, *integer-expression*)

Parameters

string-expression The string to be repeated.

integer-expression The number of times the string is to be repeated.

Example

The following statement returns the value "repeatrepeatrepeat:"

```
SELECT REPLICATE( 'repeat', 3 ) FROM iq_dummy
```

Usage

REPLICATE is the same as the REPEAT function.

Note The result data type of a REPLICATE function is a LONG VARCHAR. If you use REPLICATE in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set REPLICATE to the correct data type and size.

See "REPLACE function [String]" on page 242 for more information.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

"REPEAT function [String]" on page 241

REVERSE function [String]

Function

Takes one argument as an input of type BINARY or STRING and returns the specified string with characters listed in reverse order.

Syntax

REVERSE (*expression* | *uchar_expr*)

Parameters

expression is a character or binary-type column name, variable, or constant expression of CHAR, VARCHAR, NCHAR, NVARCHAR, BINARY, or VARBINARY type.

Example 1

```
select reverse("abcd")
----
dcba
```

Example 2

```
select reverse(0x12345000)
----------
0x00503412
```

Usage

- REVERSE, a string function, returns the reverse of expression.
- If expression is NULL, reverse returns NULL.
- Surrogate pairs are treated as indivisible and are not reversed.

Permissions

Any user can execute REVERSE.

Standards and compatibility

ANSI SQL – Compliance level: Transact-SQL extension

See also

Functions "LOWER function [String]" on page 200 and "UPPER function [String]" on page 329.

For general information about string functions, see “String functions” on page 115.

RIGHT function [String]

Function
Returns the rightmost characters of a string.

Syntax
RIGHT (*string-expression*, *numeric-expression*)

Parameters
string-expression The string to be left-truncated.

numeric-expression The number of characters at the end of the string to return.

Example
The following statement returns the value “olate:”

```
SELECT RIGHT( 'chocolate', 5 ) FROM iq_dummy
```

Usage
If the string contains multibyte characters, and the proper collation is being used, the number of bytes returned might be greater than the specified number of characters.

Note The result data type of a RIGHT function is a LONG VARCHAR. If you use RIGHT in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set RIGHT to the correct data type and size.

See “REPLACE function [String]” on page 242 for more information.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also
“LEFT function [String]” on page 195

Chapter 11, “International Languages and Character Sets” in the *System Administration Guide: Volume 1*

ROUND function [Numeric]

Function
Rounds the *numeric-expression* to the specified *integer-expression* number of places after the decimal point.

Syntax
ROUND (*numeric-expression*, *integer-expression*)

Parameters
numeric-expression The number, passed to the function, to be rounded.

integer-expression A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative expression specifies the number of significant digits to the left of the decimal point at which to round.

Examples

The following statement returns the value 123.200:

```
SELECT ROUND( 123.234, 1 ) FROM iq_dummy
```

Additional results of the ROUND function are shown in the following table:

Value	ROUND (Value)
123.4567	round (a.n,4)
123.4570	round (a.n,3)
123.4600	round (a.n,2)
123.5000	round (a.n,1)
123.0000	round (a.n, 0)
120.0000	round (a.n,-1)
100.0000	round (a.n,-2)
0.0000	round(a.n,-3)

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

"TRUNCNUM function [Numeric]" on page 268

ROWID function [Miscellaneous]

Function

Returns the internal row ID value for each row of the table.

Syntax

ROWID (*table-name*) …**FROM** *table-name*

Parameters

table-name The name of the table. Specify the name of the table within the parentheses with either no quotes or with double quotes.

Examples

The following statement returns the row ID values 1 through 10:

```
SELECT ROWID( "PRODUCTS" ) FROM PRODUCTS
```

rowid(Products)
1
2
3
.
.
.
10

The following statement returns the product ID and row ID value of all rows with a product ID value less than 400:

```
SELECT PRODUCTS.ID, ROWID ( PRODUCTS )
FROM PRODUCTS
WHERE PRODUCTS.ID < 400
```

ID	rowid(Products)
300	1
301	2
302	3

The following statement deletes all rows with row ID values greater than 50:

```
DELETE FROM PRODUCTS
WHERE ROWID ( PRODUCTS ) > 50
```

Usage

You can use the ROWID function in conjunction with other clauses to manipulate specific rows of the table.

You must specify the FROM *table-name* clause.

A limitation of the ROWID function is that it cannot use a join index of that table, eliminating any performance benefits that would normally use that join index.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

RTRIM function [String]

Function

Returns a string with trailing blanks removed.

Syntax

RTRIM (*string-expression*)

Parameters

string-expression The string to be trimmed.

Note The result data type of an RTRIM function is a LONG VARCHAR. If you use RTRIM in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set RTRIM to the correct data type and size.

See “REPLACE function [String]” on page 242 for more information.

Example

The following statement returns the string “Test Message” with all trailing blanks removed.

```
SELECT RTRIM( 'Test Message     ' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

“LTRIM function [String]” on page 201

SECOND function [Date and time]

Function

Returns a number from 0 to 59 corresponding to the second component of the given date/time value.

Syntax

SECOND (*datetime-expression*)

Parameters

datetime-expression The date/time value.

Example

The following statement returns the value 5:

```
SELECT SECOND( '1998-07-13 08:21:05' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

SECONDS function [Date and time]

Function

Returns the number of seconds since an arbitrary starting date and time, the number of seconds between two times, or adds an integer amount of seconds to a time.

Syntax

SECONDS (*datetime-expression*
| *datetime-expression*, *datetime-expression*
| *datetime-expression*, *integer-expression*)

Parameters

datetime-expression A date and time.

integer-expression The number of seconds to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of minutes are subtracted from the date/time value. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a datetime data type.

For information on casting data types, see “CAST function [Data type conversion]” on page 133.

Examples

The following statement returns the value 3600:

```
SELECT ( SECONDS( '1998-07-13 06:07:12' ) -
SECONDS( '1998-07-13 05:07:12' )) FROM iq_dummy
```

The following statement returns the value 14400, to signify the difference between the two times:

```
SELECT SECONDS( '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-05-12 21:05:12.000:

```
SELECT SECONDS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5) FROM iq_dummy
```

Usage

The second syntax returns the number of whole seconds from the first date/time to the second date/time. The number might be negative.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

SIGN function [Numeric]

Function

Returns the sign of a number.

Syntax

SIGN (*numeric-expression*)

Parameters

numeric-expression The number for which the sign is to be returned.

Example

The following statement returns the value -1:

```
SELECT SIGN( -550 ) FROM iq_dummy
```

Usage

For negative numbers, SIGN returns -1.

For zero, SIGN returns 0.

For positive numbers, SIGN returns 1.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

SIMILAR function [String]

Function

Returns an integer between 0 and 100 representing the similarity between two strings.

Syntax

SIMILAR (*string-expression1*, *string-expression2*)

Parameters

string-expression1 The first string to be compared.

string-expression2 The second string to be compared.

Example

The following statement returns the value 75:

```
SELECT SIMILAR( 'toast', 'coast' ) FROM iq_dummy
```

This signifies that the two values are 75% similar.

Usage

The function returns an integer between 0 and 100 representing the similarity between the two strings. The result can be interpreted as the percentage of characters matched between the two strings. A value of 100 indicates that the two strings are identical.

This function can be used to correct a list of names (such as customers). Some customers might have been added to the list more than once with slightly different names. Join the table to itself and produce a report of all similarities greater than 90 percent but less than 100 percent.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

SIN function [Numeric]

Function

Returns the sine of a number, expressed in radians.

Syntax

SIN (*numeric-expression*)

Parameters

numeric-expression The angle, in radians.

Example

The following statement returns the value 0.496880:

```
SELECT SIN( 0.52 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension

- **Sybase** Compatible with Adaptive Server Enterprise

See also

"ASIN function [Numeric]" on page 129

"COS function [Numeric]" on page 143

"COT function [Numeric]" on page 143

"TAN function [Numeric]" on page 267

SORTKEY function [String]

Function

Generates values that can be used to sort character strings based on alternate collation rules.

Syntax

```
SORTKEY ( string-expression
[, { collation-id
| collation-name [(collation-tailoring-string)] } ]
)
```

Parameters

string-expression The string expression must contain characters that are encoded in the character set of the database and must be STRING data type.

If *string-expression* is NULL, the SORTKEY function returns a null value. An empty string has a different sort-order value than a null string from a database column.

There is no limit on the length of the input string that the SORTKEY function can handle. The result of SORTKEY is always limited to 1024 bytes and is VARBINARY data type. If the actual results exceed 1024 bytes, the result contains only the first 1024 bytes.

collation-name A string or character variable that specifies the name of the sort order to use. You can also specify the alias char_collation, or, equivalently, db_collation, to generate sort-keys as used by the CHAR collation in use by the database.

Similarly, you can specify the alias NCHAR_COLLATION to generate sort-keys as used by the NCHAR collation in use by the database. However , Sybase IQ does not support NCHAR_COLLATION for IQ-specific objects. NCHAR_COLLATION is supported for SQL Anywhere objects on an IQ server.

collation-id A variable, integer constant, or string that specifies the ID number of the sort order to use. This parameter applies only to Adaptive Server Enterprise collations, which can be referred to by their corresponding collation ID.

If you do not specify a collation name or collation ID, the default is Default Unicode multilingual.

Valid collations are as follows:

- To see collations that are supported by SQL Anywhere, listed by label, execute `dbinit -l`.
- The Adaptive Server Enterprise collations are listed in the table below.

Description	Collation name	Collation ID
Default Unicode multilingual	default	0
CP 850 Alternative: no accent	altnoacc	39
CP 850 Alternative: lowercase first	altdict	45
CP 850 Western European: no case, preference	altnocsp	46
CP 850 Scandinavian dictionary	scandict	47
CP 850 Scandinavian: no case, preference	scannocp	48
GB Pinyin	gbpinyin	n/a
Binary sort	binary	50
Latin-1 English, French, German dictionary	dict	51
Latin-1 English, French, German no case	nocase	52
Latin-1 English, French, German no case, preference	nocasep	53
Latin-1 English, French, German no accent	noaccent	54
Latin-1 Spanish dictionary	espdict	55
Latin-1 Spanish no case	espnocs	56
Latin-1 Spanish no accent	espnoac	57
ISO 8859-5 Russian dictionary	rusdict	58
ISO 8859-5 Russian no case	rusnocs	59
ISO 8859-5 Cyrillic dictionary	cyrdict	63
ISO 8859-5 Cyrillic no case	cyrnocs	64
ISO 8859-7 Greek dictionary	elldict	65
ISO 8859-2 Hungarian dictionary	hundict	69
ISO 8859-2 Hungarian no accents	hunnoac	70
ISO 8859-2 Hungarian no case	hunnocs	71
ISO 8859-5 Turkish dictionary	turdict	72
ISO 8859-5 Turkish no accents	turnoac	73
ISO 8859-5 Turkish no case	turnocs	74
CP 874 (TIS 620) Royal Thai dictionary	thaidict	1

Description	Collation name	Collation ID
ISO 14651 ordering standard	14651	22
Shift-JIS binary order	sjisbin	179
Unicode UTF-8 binary sort	utf8bin	24
EUC JIS binary order	eucjisbn	192
GB2312 binary order	gb2312bn	137
CP932 MS binary order	cp932bin	129
Big5 binary order	big5bin	194
EUC KSC binary order	euckscbn	161

collation-tailoring-string (Optional) Specify collation tailoring options (*collation-tailoring-string*) for additional control over sorting and comparison of characters. These options take the form of keyword=value pairs assembled in parentheses, following the collation name. For example,

```
'UCA(locale=es;case=LowerFirst;accent=respect)'
```

The syntax for specifying these options is identical to the COLLATION clause of the CREATE DATABASE statement. See *collation-tailoring-string* in "CREATE DATABASE statement" in Chapter 1, "SQL Statements" of *Reference: Statements and Options*.

Note All of the collation tailoring options are supported for SQL Anywhere databases, when specifying the Unicode Collation Algorithm (UCA) collation. For all other collations, only case sensitivity tailoring is supported.

Example

The following statement queries the Employees table and returns the FirstName and Surname of all employees, sorted by the sort-key values for the Surname column using the dict collation (Latin-1, English, French, German dictionary):

```
SELECT Surname, GivenName FROM Employees ORDER BY
SORTKEY( Surname, 'dict' );
```

Usage

The SORTKEY function generates values that can be used to order results based on predefined sort order behavior. This allows you to work with character sort order behaviors that may not be available from the database collation. The returned value is a binary value that contains coded sort order information for the input string that is retained from the SORTKEY function.

For example, you can store the values returned by the SORTKEY function in a column with the source character string. The following SELECT statement retrieves data from table T1 in the sorted order of c1 according to the Thai dictionary:

```
SELECT rid, c1 from T1 ORDER BY SORTKEY(c1)
```

You instead store the value returned by SORTKEY in a column with the source character string. To retrieve the character data in the required order, the SELECT statement needs to include only an ORDER BY clause on the column that contains the results of running the SORTKEY function:

```
UPDATE T1 SET shadowc1=SORTKEY(c1) FROM T1;
SELECT rid, c1 FROM T1 ORDER BY shadowc1
```

The SORTKEY function guarantees that the values it returns for a given set of sort order criteria work for the binary comparisons that are performed on VARBINARY data types.

Generating sort-keys for queries can be expensive. As an alternative for frequently requested sort-keys, consider creating a computed column to hold the sort-key values, and then referencing that column in the ORDER BY clause of the query.

With respect to collation tailoring, full sensitivity is generally the intent when creating sort-keys, so when you specify a non-UCA collation, the default tailoring applied is equivalent to case=Respect. For example, the following two statements are equivalent:

```
SELECT SORTKEY( 'abc', '1252LATIN1' );
SELECT SORTKEY( 'abc', '1252LATIN1(case=Respect)' );
```

When specifying a non-UCA collation, by default, collation tailorings are accent and case-sensitive. However, for non-UCA collations, you can override only the case sensitivity using a collation tailoring. For example:

```
SELECT SORTKEY( 'abc', '1252LATIN1(case=LowerFirst)' );
```

If the database was created without specifying tailoring options, the following two clauses may generate different sort orders, even if the database collation name is specified for the SORTKEY function:

```
ORDER BY string-expression
ORDER BY SORTKEY( string-expression, database-
collation-name )
```

Different sort orders may be generated, because the default tailoring settings used for database creation and for the SORTKEY function are different. To get the same behavior from SORTKEY as for the database collation, either provide a tailoring syntax for *collation-tailoring-string* that matches the settings for the database collation, or specify db_collation for collation-name. For example:

```
SORTKEY( expression, 'db_collation' )
```

For information on using collation tailoring and the SORTKEY function with a SQL Anywhere database, see "SQL Functions" in the *SQL Anywhere Server – SQL Reference*.

Note Sort-key values created using a version of Sybase IQ earlier than 15.0 do not contain the same values created using version 15.0 and later. This may be a problem for your applications if your pre-15.0 database has sort-key values stored within it, especially if sort-key value comparison is required by your application. Regenerate any sort-key values in your database that were generated using a version of Sybase IQ earlier than 15.0.

See also

"SORT_COLLATION option" in Chapter 2, "Database Options"

"String functions" on page 115

Chapter 11, "International Languages and Character Sets" in *System Administration Guide: Volume 1*

SOUNDEX function [String]

Function

Returns a number representing the sound of a string.

Syntax

SOUNDEX (*string-expression*)

Parameters

string-expression The string.

Example

The following statement returns two numbers, representing the sound of each name. The SOUNDEX value for each argument is 3827.

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' ) FROM
iq_dummy
```

SOUNDEX ('Smith') is equal to SOUNDEX ('Smythe').

Usage

The SOUNDEX function value for a string is based on the first letter and the next three consonants other than H, Y, and W. Doubled letters are counted as one letter. For example:

```
SOUNDEX( 'apples' ) FROM iq_dummy
```

is based on the letters A, P, L, and S.

Multibyte characters are ignored by the SOUNDEX function.

Although it is not perfect, SOUNDEX normally returns the same number for words that sound similar and that start with the same letter.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise, except that Adaptive Server Enterprise returns a CHAR(4) result and Sybase IQ returns an integer

SPACE function [String]

Function

Returns a specified number of spaces.

Syntax

SPACE (*integer-expression*)

Parameters

integer-expression The number of spaces to return.

Example

The following statement returns a string containing 10 spaces:

```
SELECT SPACE( 10 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

SQRT function [Numeric]

Function

Returns the square root of a number.

Syntax

SQRT (*numeric-expression*)

Parameters

numeric-expression The number for which the square root is to be calculated.

Example

The following statement returns the value 3:

```
SELECT SQRT( 9 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

SQUARE function [Numeric]

Function
Returns the square of the specified expression as a float.

Syntax
SQUARE (*numeric-expression*)

Parameters
expression Is a column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, the SQUARE function generates an error. The return value is of DOUBLE data type.

Usage
SQUARE function takes one argument. For example, SQUARE (*12.01*) returns 144.240100.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

STDDEV function [Aggregate]

Function
Returns the standard deviation of a set of numbers.

Syntax
STDDEV ([ALL] *expression*)

Parameters
expression Any numeric data type (FLOAT, REAL, or DOUBLE precision) expression.

Examples
Given this data:

```
SELECT Salary FROM Employees WHERE DepartmentID = 300
```

Salary
51432.000
57090.000
42300.000
43700.00
36500.000
138948.000
31200.000
58930.00
75400.00

The following statement returns the value 32617.8446712838471:

```
SELECT STDDEV ( Salary ) FROM Employees
WHERE DepartmentID = 300
```

Given this data:

```
SELECT UnitPrice FROM Products WHERE Name = 'Tee Shirt'
```

Name	**UnitPrice**
Tee Shirt	9.00
Tee Shirt	14.00
Tee Shirt	14.00

The following statement returns the value 2.88675134594813049:

```
SELECT STDDEV ( UnitPrice ) FROM Products
WHERE Name = 'Tee Shirt'
```

Usage

The formula used to calculate STDDEV is:

$$stddev = \sqrt{variance}$$

STDDEV returns a result of data type DOUBLE precision floating-point. If applied to the empty set, the result is NULL, which returns NULL for a one-element input set.

STDDEV does not support the keyword DISTINCT. A syntax error is returned if you use DISTINCT with STDDEV.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

“STDDEV_SAMP function [Aggregate]” on page 259

“VARIANCE function [Aggregate]” on page 333

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

STDDEV_POP function [Aggregate]

Function

Computes the standard deviation of a population consisting of a numeric-expression, as a DOUBLE.

Syntax

STDDEV_POP ([ALL] *expression*)

Parameters

expression The expression (commonly a column name) whose population-based standard deviation is calculated over a set of rows.

Examples

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year( ship_date ) AS Year, quarter( ship_date )
   AS Quarter, AVG( quantity ) AS Average,
   STDDEV_POP ( quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
   ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	14.2794
2000	2	27.050847	15.0270
...	...	...	...

Usage

Computes the population standard deviation of the provided *value expression* evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the population variance.

$$\sqrt{\frac{\sum (x_i - \bar{x})^2}{n}}$$

Standards and compatibility

- **SQL99** SQL/foundation feature outside of core SQL.
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"Analytical functions" on page 104

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

STDDEV_SAMP function [Aggregate]

Function

Computes the standard deviation of a sample consisting of a numeric-expression, as a DOUBLE.

Note STDDEV_SAMP is an alias for STDDEV.

Syntax

STDDEV_SAMP ([ALL] *expression*)

Parameters

expression The expression (commonly a column name) whose sample-based standard deviation is calculated over a set of rows.

Examples

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year( ship_date ) AS Year, quarter( ship_date )
   AS Quarter, AVG( quantity ) AS Average,
   STDDEV_SAMP( quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
   ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	14.3218
2000	2	27.050847	15.0696
...	...	...	...

Usage

Computes the sample standard deviation of the provided *value expression* evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the sample variance.

NULL returns NULL for a one-element input set.

Standard deviations are computed according to the following formula, which assumes a normal distribution:

$$\sqrt{\frac{\sum (x_i - \bar{x})^2}{(n-1)}}$$

Standards and compatibility

- **SQL99** SQL/foundation feature outside of core SQL.
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"Analytical functions" on page 104

"STDDEV function [Aggregate]" on page 257

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

STR function [String]

Function

Returns the string equivalent of a number.

Syntax

STR (*numeric-expression* [, *length* [, *decimal*]])

Parameters

numeric-expression Any approximate numeric (FLOAT, REAL, or DOUBLE precision) expression.

length The number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, the sign, if any, and blanks). The default is 10 and the maximum length is 255.

decimal The number of digits to the right of the decimal point to be returned. The default is 0.

Examples

The following statement returns a string of six spaces followed by 1234, for a total of ten characters:

```
SELECT STR( 1234.56 ) FROM iq_dummy
```

The following statement returns the result 1234.5:

```
SELECT STR( 1234.56, 6, 1 ) FROM iq_dummy
```

Usage

If the integer portion of the number cannot fit in the length specified, then the result is NULL. For example, the following statement returns NULL:

```
SELECT STR( 1234.56, 3 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

STR_REPLACE function [String]

Function

Takes three arguments as input of type BINARY or STRING and replaces any instances of the second string expression (*string_expr2*) that occur within the first string expression (*string_expr1*) with a third expression (*string_expr3*).

STR_REPLACE is an alias of REPLACE function

Syntax

REPLACE (*string_expr1*, *string_expr2*, *string_expr3*)

Parameters

string_expr1 is the source string, or the string expression to be searched, expressed as CHAR, VARCHAR, UNICHAR, UNIVARCHAR, VARBINARY, or BINARY data type.

string_expr2 is the pattern string, or the string expression to find within the first expression (*string_expr1*) and is expressed as CHAR, VARCHAR, UNICHAR, UNIVARCHAR, VARBINARY, or BINARY data type.

string_expr3 is the replacement string expression, expressed as CHAR, VARCHAR, UNICHAR, UNIVARCHAR, VARBINARY, or BINARY data type.

Example 1

Replaces the string *def* within the string *cdefghi* with *yyy*.

```
select replace("cdefghi", "def", "yyy")
-------------
```

```
cyyyghi
(1 row(s) affected)
```

Example 2

Replaces all spaces with “toyota”

```
select str_replace ("chevy, ford, mercedes",
"","toyota")
----------
chevy,toyotaford,toyotamercedes
(1 row(s) affected)
```

Example 3

Accepts NULL in the third parameter and treats it as an attempt to replace *string_expr2* with NULL, effectively turning STR_REPLACE into a “string cut” operation. Returns “abcghijklm”:

```
select str_replace("abcdefghijklm", "def", NULL)
----------
abcghijklm
(1 row affected)
```

Usage

- Takes any data type as input and returns STRING or BINARY.

  For example, an empty string passed as an argument (“”) is replaced with one space (“ ”) before further evaluation occurs. This is true for both BINARY and STRING types.
- All arguments can have a combination of BINARY and STRING data types.
- The result length may vary, depending upon what is known about the argument values when the expression is compiled. If all arguments are columns or host variables assigned to constants, Sybase IQ calculates the result length as:

  ```
  result_length = ((s/p)*(r-p)+s)
  WHERE
     s = length of source string
     p = length of pattern string
     r = length of replacement string
  IF (r-p) <= 0, result length = s
  ```
- If Sybase IQ cannot calculate the result length because the argument values are unknown when the expression is compiled, the result length used is 255.
- RESULT_LEN never exceeds 32767.

Permissions

Any user can execute STR_REPLACE.

Standards and compatibility

ANSI SQL – Compliance level: Transact-SQL extension

See also

Data types CHAR, VARCHAR, UNICHAR, UNIVARCHAR, VARBINARY, or BINARY. See Chapter 3, "SQL Data Types."

Function "LENGTH function [String]" on page 197.

For general information about string functions, see "String functions" on page 115.

STRING function [String]

Function

Concatenates one or more strings into one large string.

Syntax

STRING (*string-expression* [, …])

Parameters

string-expression A string.

If only one argument is supplied, it is converted into a single expression. If more than one argument is supplied, they are concatenated into a single string.

A NULL is treated as an empty string ('').

Example

The following statement returns the value testing123:

```
SELECT STRING( 'testing', NULL, 123 )
FROM iq_dummy
```

Usage

Numeric or date parameters are converted to strings before concatenation. You can also use the STRING function to convert any single expression to a string by supplying that expression as the only parameter.

If all parameters are NULL, STRING returns NULL.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

STRTOUUID function [String]

Function

Converts a string value to a unique identifier (UUID or GUID) value.

Syntax

STRTOUUID (*string-expression*)

Parameters

string-expression A string in the format *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*

Example

```
CREATE TABLE T (
  pk uniqueidentifier primary key,
  c1 int);
```

```
INSERT INTO T (pk, c1)
  VALUES (STRTOUUID
  ('12345678-1234-5678-9012-123456789012'), 1);
```

Usage

Converts a string in the format *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* where *x* is a hexadecimal digit, to a unique identifier value. If the string is not a valid UUID string, NULL is returned.

You can use STRTOUUID to insert UUID values into a Sybase IQ database.

Standards and compatibility

- **SQL92** Vendor extension
- **SQL99** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"NEWID function [Miscellaneous]" on page 208

"UUIDTOSTR function [String]" on page 331

UNIQUEIDENTIFIER in "Binary data types" on page 77

STUFF function [String]

Function

Deletes a number of characters from one string and replaces them with another string.

Syntax

STUFF (*string-expression1*, *start*, *length*, *string-expression2*)

Parameters

string-expression1 The string to be modified by the STUFF function.

start The character position at which to begin deleting characters. The first character in the string is position 1.

length The number of characters to delete.

string-expression2 The string to be inserted.

Example

The following statement returns the value "chocolate pie":

```
SELECT STUFF( 'chocolate cake', 11, 4, 'pie' )
FROM iq_dummy
```

Usage

To delete a portion of a string using STUFF, use a replacement string of NULL. To insert a string using STUFF, use a length of zero.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

"INSERTSTR function [String]" on page 189

SUBSTRING function [String]

Function
Returns a substring of a string.

Syntax
{ **SUBSTRING** | **SUBSTR** } (*string-expression*, *start* [, *length*])

Parameters
string-expression The string from which a substring is to be returned.

start The start position of the substring to return, in characters. A negative starting position specifies a number of characters from the end of the string instead of the beginning. The first character in the string is at position 1.

length The length of the substring to return, in characters. A positive *length* specifies that the substring ends *length* characters to the right of the starting position, while a negative *length* specifies that the substring ends *length* characters to the left of the starting position.

Examples
The following statement returns "back":

```
SELECT SUBSTRING ( 'back yard', 1 , 4 )
FROM iq_dummy
```

The following statement returns yard:

```
SELECT SUBSTR ( 'back yard', -1 , -4 )
FROM iq_dummy
```

The following statement returns 0x2233:

```
SELECT SUBSTR ( 0x112233445566, 2, 2 )
FROM iq_dummy
```

Usage
If *length* is specified, the substring is restricted to that length. If no length is specified, the remainder of the string is returned, starting at the *start* position.

Both *start* and *length* can be negative. Using appropriate combinations of negative and positive numbers, you can get a substring from either the beginning or end of the string.

Standards and compatibility
- **SQL92** Vendor extension
- **Sybase** SUBSTR is not supported by Adaptive Server Enterprise. Use SUBSTRING instead.

SUM function [Aggregate]

Function
Returns the total of the specified expression for each group of rows.

Syntax
SUM (*expression* | **DISTINCT** *column-name*)

Parameters
expression The object to be summed. This is commonly a column name.

DISTINCT column-name Computes the sum of the unique values in *column-name* for each group of rows. This is of limited usefulness, but is included for completeness.

Example

The following statement returns the value 3749146.740:

```
SELECT SUM( salary )
FROM Employees
```

Usage

Rows where the specified expression is NULL are not included.

Returns NULL for a group containing no rows.

Standards and compatibility

- **SQL92** SQL92 compatible.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

“AVG function [Aggregate]” on page 131

“COUNT function [Aggregate]” on page 146

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

SUSER_ID function [System]

Function

Returns an integer user identification number.

Syntax

SUSER_ID ([*user-name*])

Parameters

user-name The user name.

Examples

The following statement returns the user identification number 1:

```
SELECT SUSER_ID ('DBA') FROM iq_dummy
```

The following statement returns the user identification number 0:

```
SELECT SUSER_ID ('SYS') FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ

See also

“SUSER_NAME function [System]” on page 266

“USER_ID function [System]” on page 330

SUSER_NAME function [System]

Function

Returns the user name.

Syntax **SUSER_NAME** ([*user-id*])

Parameters **user-id** The user identification number.

Examples The following statement returns the value DBA:

```
SELECT SUSER_NAME ( 1 ) FROM iq_dummy
```

The following statement returns the value SYS:

```
SELECT SUSER_NAME ( 0 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ. In Adapter Server Enterprise, SUSER_NAME returns the server user name.

See also “SUSER_ID function [System]” on page 266

“USER_NAME function [System]” on page 330

TAN function [Numeric]

Function Returns the tangent of a number.

Syntax **TAN** (*numeric-expression*)

Parameters **numeric-expression** An angle, in radians.

Example The following statement returns the value 0.572561:

```
SELECT TAN( 0.52 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also “COS function [Numeric]” on page 143

“SIN function [Numeric]” on page 250

TODAY function [Date and time]

Function Returns the current date. This is the historical syntax for CURRENT DATE.

Syntax **TODAY** (*)

Example The following statement returns the current day according to the system clock.

```
SELECT TODAY( * ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

TRIM function [String]

Function

Removes leading and trailing blanks from a string.

Syntax

TRIM (*string-expression*)

Parameters

string-expression The string to be trimmed.

Note The result data type of a TRIM function is a LONG VARCHAR. If you use TRIM in a SELECT INTO statement, you must have a Large Objects Management option license, or use CAST and set TRIM to the correct data type and size.

See “REPLACE function [String]” on page 242 for more information.

Example

The following statement returns the value “chocolate” with no leading or trailing blanks.

```
SELECT TRIM( '   chocolate   ' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise. Use LTRIM and RTRIM instead.

See also

“LTRIM function [String]” on page 201

“RTRIM function [String]” on page 247

TRUNCNUM function [Numeric]

Function

Truncates a number at a specified number of places after the decimal point.

Syntax

TRUNCNUM (*numeric-expression*, *integer-expression*)

Parameters

numeric-expression The number to be truncated.

integer-expression A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative expression specifies the number of significant digits to the left of the decimal point at which to round.

Examples

The following statement returns the value 600:

```
SELECT TRUNCNUM( 655, -2 ) FROM iq_dummy
```

The following statement: returns the value 655.340:

```
SELECT TRUNCNUM( 655.348, 2 ) FROM iq_dummy
```

Usage

This function is the same as TRUNCATE, but does not cause keyword conflicts.

You can use combinations of ROUND, FLOOR, and CEILING to provide similar functionality.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported in Adaptive Server Enterprise

See also

"ROUND function [Numeric]" on page 245

TS_ARMA_AR function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Calculates the least-square estimates of parameters for an autoregressive moving average (ARMA) model, and returns the requested autoregressive estimate.

Syntax

TS_ARMA_AR (*timeseries_expression*, *ar_count*, *ar_elem*, *method*)

OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

ar_count An integer containing the number of autoregressive values to compute.

ar_elem An integer identifying the element in the computed AR array that should be returned. *ar_elem* must be greater than 0 and less than or equal to *ar_count*.

method (Optional) An integer that identifies the type of procedure used to compute estimates. 0 (the default value) = method of least squares; 1 = method of moments.

window-spec TS_ARMA_AR is an OLAP function requiring an OVER () clause.

Usage

TS_ARMA_AR time series function returns a double-precision floating-point value containing the autoregressive estimate. TS_ARMA_AR calls the function imsls_d_arma in the IMSL libraries.

IMSL mapping

The arguments of TS_ARMA_AR map to the IMSL library function imsls_d_arma as follows:

```
params = imsls_d_arma(n_objs, z, p, q, methodID, 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

p Maps to the user-defined aggregate function argument *ar_count*.

q =1.

methodID Maps to the *method* argument of TS_ARMA_AR. Can be set to either IMSLS_METHOD_OF_MOMENTS or IMSLS_LEAST_SQUARES.

For detailed information on how imsls_d_arma performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_ARMA_AR function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data.

Table 4-27: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381
5	2.77555
6	2.73657
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859

rownum	data
15	0.833405
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422
42	4.11499
43	2.77188
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns the first element of an autoregressive estimate consisting of one value from the data column using the method of least squares:

```
SELECT TS_ARMA_AR(data,1,1,0) OVER (ORDER BY rownum
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-28: Values returned from TS_ARMA_AR

res
0.898793
0.898793
0.898793
0.898793
0.898793
0.898793
0.898793
0.898793
0.898793
...
0.898793

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library

TS_ARMA_CONST function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Calculates the least-square estimates of parameters for an autoregressive moving average (ARMA) model, and returns an estimated constant.

Syntax

TS_ARMA_CONST (*timeseries_expression*, *method*)
OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

method An integer that identifies the type of procedure used to compute estimates. 0 (the default value) = method of least squares; 1 = method of moments.

window-spec TS_ARMA_CONST is an OLAP function requiring an OVER () clause.

Usage

This time series function returns a double-precision floating-point value containing the constant estimate produced by the function. TS_ARMA_CONST calls the function imsls_d_arma in the IMSL libraries.

IMSL mapping

The arguments of TS_ARMA_CONST map to the IMSL library function imsls_d_arma as follows:

```
params = imsls_d_arma(n_objs, z, p, q, IMSLS_CONSTANT,
method_id, 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

p =1.

q =1.

MethodID Maps to the *method* argument of TS_ARMA_CONST.

For detailed information on how the function imsls_d_arma performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example 1

This example shows an input data table, a SQL statement containing the TS_ARMA_CONST function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-29: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381

rownum	data
5	2.77555
6	2.73657
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422

rownum	data
42	4.11499
43	2.77188
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns an estimated constant from the data column using the method of least squares:

```
SELECT TS_ARMA_CONST(data,0) OVER (ORDER BY ROWNUM rows
BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS
res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-30: Values returned from TS_ARMA_CONST Example 1

res
0.082077
0.082077
0.082077
0.082077
0.082077
0.082077
0.082077
0.082077
0.082077
...
0.082077

Example 2

This example provides a sample query that returns estimates for the AR, MA, and constant parameters. The first element for AR and MA in the array contains one element. See Table 4-29 on page 273 for the DATASET table.

```
SELECT TS_ARMA_AR(DATA,1,1,0) OVER (ORDER BY rownum
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS ar_param, ts_arma_ma(data,1,1,0) OVER
```

```
(ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND
UNBOUNDED FOLLOWING) AS ma_param, ts_arma_const(data,0)
OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING
AND UNBOUNDED FOLLOWING) AS const_param FROM DATASET
```

Sybase IQ returns 50 rows of data, each containing the same three values:

Table 4-31: Values returned from TS_ARMA_CONST Example 2

ar_param	ma_param	const_param
0.898793	0.105075	0.082077
0.898793	0.105075	0.082077
0.898793	0.105075	0.082077
0.898793	0.105075	0.082077
0.898793	0.105075	0.082077
0.898793	0.105075	0.082077
0.898793	0.105075	0.082077
0.898793	0.105075	0.082077
0.898793	0.105075	0.082077
...	...	...
0.898793	0.105075	0.082077

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library

TS_ARMA_MA function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Calculates the least-square estimates of parameters for an autoregressive moving average (ARMA) model, and returns the requested moving average estimate.

Syntax

TS_ARMA_MA (*timeseries_expression*, *ma_count*, *ma_elem*, *method*)

OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

ma_count An integer containing the number of autoregressive values to compute.

ma_elem An integer identifying the element to return from the computed moving average array. The integer must be greater than 0 and less than or equal to *ma_count*.

method (Optional) An integer identifying the procedure to use to calculate estimates. 0 (the default value) = method of least squares and 1 = method of moments.

window-spec TS_ARMA_MA is an OLAP function requiring an OVER () clause.

Usage

This time series function returns a double-precision floating-point value representing the moving average estimate. TS_ARMA_MA calls the function imsls_d_arma in the IMSL libraries.

IMSL mapping

The arguments of TS_ARMA_MA map to the IMSL library function imsls_d_arma as follows:

```
params = imsls_d_arma(n_objs, z, p, q, method_id, 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

p =1.

q Maps to the user-defined aggregate function argument *ma_count*.

method_id Maps to the *method* argument of TS_ARMA_MA.

For detailed information on how the function imsls_d_arma performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_ARMA_MA function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-32: Input data table DATASET

rownum	data
1	0.315523
2	0.485859

rownum	data
3	0.676886
4	1.97381
5	2.77555
6	2.73657
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696

rownum	data
40	2.80171
41	3.78422
42	4.11499
43	2.77188
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns the first element of an array containing one element from the data column using the method of least squares:

```
SELECT TS_ARMA_MA(data,1,1,0) OVER (ORDER BY rownum
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-33: Values returned from TS_ARMA_MA

res
0.105075
0.105075
0.105075
0.105075
0.105075
0.105075
0.105075
0.105075
0.105075
...
0.105075

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library

TS_AUTOCORRELATION function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Calculates the sample autocorrelation function of a stationary time series.

Syntax

TS_AUTOCORRELATION (*timeseries_expression*, *lagmax*, *lag_elem*)
OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

lagmax An integer specifying the maximum lag of autocovariances, autocorrelations, and standard errors of autocorrelations. The integer must be greater than or equal to 1, and less than the number of elements in the time series.

lag_elem An integer specifying which element in the autocorrelation array is to be returned. The integer must be greater than zero, and less than or equal to *lagmax*.

window-spec TS_AUTOCORRELATION is an OLAP function requiring an OVER () clause.

Usage

This time series function returns a double-precision floating-point value representing the autocorrelation value. TS_AUTOCORRELATION calls the function imsls_d_autocorrelation in the IMSL libraries.

IMSL mapping

The arguments of TS_AUTOCORRELATION map to the IMSL library function imsls_d_autocorrelation() as follows:

```
params = imsls_d_autocorrelation(n_objs, x[], lagmax,
0);
```

n_objs Contains the number of rows in the current window frame.

x[] Contains the value of *timeseries_expression* for the current window frame.

lagmax Maps to the user-defined aggregate function argument *lag_max*.

For detailed information on how the function imsls_d_autocorrelation performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_AUTOCORRELATION function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-34: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381
5	2.77555
6	2.73657
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423

rownum	data
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422
42	4.11499
43	2.77188
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns the second element from an array containing autocorrelations of the time series data from the data column:

```
SELECT TS_AUTOCORRELATION(data,2,2) OVER (ORDER BY
ROWNUM rows BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-35: Values returned from TS_AUTOCORRELATION

res
0.803659
0.803659
0.803659
0.803659
0.803659
0.803659

res
0.803659
0.803659
0.803659
...
0.803659

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library

TS_AUTO_UNI_AR function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Performs automatic selection and fitting of a univariate autoregressive time series model.

Syntax

TS_AUTO_UNI_AR (*timeseries_expression*, *ar_count*, *ar_elem*, *method*)
OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

ar_count An integer containing the number of autoregressive values to compute.

ar_elem An integer identifying which computed autoregressive value to return. The integer must be greater than zero, and less than or equal to *ar_count*.

method (Optional) An integer identifying which method to use when computing AR coefficients, where:
0 = method of moments
1 = method of least squares (the default value)
2 = maximum likelihood.

window-spec TS_AUTO_UNI_AR is an OLAP function requiring an OVER () clause.

Usage

This time series function returns a double-precision floating-point number containing the autoregressive estimate. TS_AUTO_UNI_AR calls the function imsls_d_auto_uni_ar in the IMSL libraries.

IMSL mapping

The arguments of TS_AUTO_UNI_AR map to the IMSL library function imsls_d_auto_uni_ar as follows:

```
params = imsls_d_auto_uni_ar (n_objs, z[], maxlag, p,
method, 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

maxlag Maps to the user-defined aggregate function argument *ar_count*.

p The output parameter, representing the number of autoregressive parameters in the model with minimum AIC.

method Maps to the user-defined aggregate function argument *method*. If *ar_elem* is greater than *p*, and if your IMSL library time series function error-handling value is set to 0, IQ returns a null. If your IMSL library time series function error-handling value is set to a value other than 0, IQ displays an error message indicating that *ar_elem* is greater than *p*. See "IMSL library time series function error-handling" on page 125.

For detailed information on how the function imsls_d_auto_uni_ar performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_AUTO_UNI_AR function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-36: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381
5	2.77555
6	2.73657

rownum	data
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422
42	4.11499
43	2.77188

rownum	data
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns the first element from an array containing two elements from the data column:

```
SELECT TS_AUTO_UNI_AR(data,2,1,0) OVER (ORDER BY ROWNUM
rows BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-37: Values returned from TS_AUTO_UNI_AR

res
0.883453
0.883453
0.883453
0.883453
0.883453
0.883453
0.883453
0.883453
0.883453
...
0.883453

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library

TS_BOX_COX_XFORM function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Performs a forward or inverse Box-Cox power transformation.

Syntax

TS_BOX_COX_XFORM (*timeseries_expression*, *power* [, *shift* [, *inverse*]]) **OVER** (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

power A double-precision floating-point value representing an exponent parameter in the Box-Cox power transformation.

shift (Optional) A double-precision floating-point value representing a shift parameter. The value must satisfy the relation:
min(timeseries)+shift>0

shift defaults to 0.0.

inverse (Optional) A tinyint value; if set to 1, IQ performs an inverse transformation. If 0 or null, IQ performs a forward transformation. The default value is 0.

window-spec TS_BOX_COX_XFORM is an OLAP function requiring an OVER () clause with an unbounded window. TS_BOX_COX_XFORM does not support value-based windows; for example, you cannot use a range specifier in the OVER () clause.

Usage

TS_BOX_COX_XFORM returns the corresponding calculated transformed value for each element in the time series; it calls the function imsls_d_box_cox_transform in the IMSL libraries.

IMSL mapping

The arguments of TS_BOX_COX_XFORM map to the IMSL library function imsls_d_box_cox_transform as follows:

```
params = imsls_d_box_cox_transform(n_objs, z[], power,
IMSLS_SHIFT, shift [, IMSLS_INVERSE], 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

power Maps to the user-defined aggregate function argument *power*.

shift Maps to the user-defined aggregate function argument *shift*.

IMSLS_INVERSE If the user-defined aggregate function argument *inverse* is 1, IQ calls the Box-Cox transform with IMSLS_INVERSE, otherwise this argument is left out of the function call.

For detailed information on how the function imsls_d_box_cox_transform performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_BOX_COX_XFORM function, and the data values returned by the function. This example uses the following table (called BOX_COX_XFORM_DATASET) as its input data. The BOX_COX_XFORM_DATASET table contains 13 rows of time series data:

Table 4-38: Input data table BOX_COX_XFORM_DATASET

rownum	data
1	7
2	26
3	6
4	60
5	78.5
6	1
7	29
8	15
9	52
10	74.3
11	11
12	56
13	8

The following SQL statement returns the Box-Cox power transformation from the data column:

```
SELECT TS_BOX_COX_XFORM(data,1.0,1.0,0) OVER (ORDER BY
rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM BOX_COX_XFORM_DATASET
```

Sybase IQ returns the following 13 rows:

Table 4-39: Values returned from TS_BOX_COX_XFORM

res
8
27
7

res
61
79.5
2
30
16
53
75.3
12
57
9

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library

TS_DIFFERENCE function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Differences a seasonal or nonseasonal time series.

Syntax

TS_DIFFERENCE (*timeseries_expression*, *period1* [, *period2* [, ...*period 10*]]) **OVER** (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series to be differenced.

period1 ... period10 Each period is an integer expression containing the period in which the time series is to be differenced. You must specify at least one period, and you can specify up to 10 periods.

window-spec TS_DIFFERENCE is an OLAP function requiring an OVER () clause with an unbounded window. This function does not support value-based windows; for example, you cannot use a range specifier in the OVER () clause.

Usage

For each element in the time series, TS_DIFFERENCE returns the corresponding calculated differenced value for the time series; it calls the function imsls_d_difference in the IMSL libraries.

IMSL mapping

The arguments of TS_DIFFERENCE map to the IMSL library function imsls_d_difference as follows:

```
params = imsls_d_difference(n_objs, z[], n_differences,
periods [], 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

n_differences Maps to the *period* arguments defined in TS_DIFFERENCE.

period An array of the *period* arguments defined in TS_DIFFERENCE.

For detailed information on how the function imsls_d_difference performs time series calculations, see *IMSL C Numerical Library User's Guide Volume 2 of 2: C Stat Library.*

Example

This example shows an input data table, a SQL statement containing the TS_DIFFERENCE function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-40: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381
5	2.77555
6	2.73657
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405

rownum	data
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422
42	4.11499
43	2.77188
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement differences data from the data column:

```
SELECT TS_DIFFERENCE(data,1) OVER (ORDER BY ROWNUM rows
BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS
res FROM DATASET
```

Sybase IQ returns 50 rows:

Table 4-41: Values returned from TS_DIFFERENCE

res
NULL
0.170336
0.191027
1.29692
0.801743
-0.038988
-0.09424
1.61886
-1.12477
1.02925
-1.20614
-0.814478
-0.157049
-1.18213
0.027546
1.45734
-0.990302
-0.833325
-0.637229
-0.08655
-0.12594
-0.122914
-1.39596
0.919785
-0.449474
0.037273
-0.954345
-0.562983
1.98379
0.88304
-0.345265
0.934656
0.069088
-0.249428
0.795766
-1.8145

res
1.27016
1.39266
-0.141794
0.934752
0.982506
0.330772
-1.34311
1.23124
0.209869
0.791146
-0.259155
0.15124
-0.963484
0.383186

Note The first row of results is NULL because the IMSL library returned a not a number (NaN) value for that row.

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library

TS_ESTIMATE_MISSING function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Estimates the missing values in a time series and returns them as a new time series, interspersed with the original time series.

Syntax

TS_ESTIMATE_MISSING (*timeseries_expression*, *method*)

OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series to be differenced. If a null-value is provided, it is assumed to reflect a gap in the time series, the value of which will be computed by the function.

method (Optional) An integer specifying the method to use when determining missing values:

- **0** (default) — estimates the missing time series observations in a gap by the median of the last four time series values before and the first four values after the gap.
- **1**— uses a cubic spline interpolation method to estimate missing values. Here, the interpolation is again done over the last four time series values before and the first four values after the gap.
- **2** — assumes that the time series before the gap can be well described by an AR(1) process.
- **3** — uses an AR(p) model to estimate missing values by a one-step-ahead forecast.

window-spec TS_ESTIMATE_MISSING is an OLAP function requiring an OVER () clause with an unbounded window. This function does not support value-based windows; for example, you cannot use a range specifier in the OVER () clause.

Usage

Use TS_ESTIMATE_MISSING to estimate any missing equidistant time points using one of the four estimation methods. TS_ESTIMATE_MISSING calls the function imsls_d_estimate_missing in the IMSL libraries

You cannot use TS_ESTIMATE_MISSING if more than two consecutive NULL values exist in your set of timepoints. If the first or last two values in the set of timepoints are NULL, the function returns NULL.

IMSL mapping

The arguments of TS_ESTIMATE_MISSING map to the IMSL library function imsls_d_estimate_missing as follows:

```
params = imsls_d_estimate_missing(n_objs, tpoints[],
z[], method, 0);
```

n_objs Contains the number of rows in the current window frame.

tpoints An array of indexes for specifying missing values in a sequence of timepoints.

z[] The accumulated *timeseries_expression*, obtained during calls to next_value.

method Maps to the *method* argument defined in TS_ESTIMATE_MISSING.

For detailed information on how the function imsls_d_estimate_missing performs time series calculations, see *IIMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_ESTIMATE_MISSING function, and the data values returned by the function. This example uses the following table (called EST_MISSING_DATASET) as its input data. The EST_MISSING_DATASET table contains nine rows of time series data:

Table 4-42: Input data table EST_MISSING_DATASET

rownum	data
1	2.8223
2	-0.5721
3	2.2771
4	NULL
5	1.2648
6	1.0278
7	0.6991
8	-1.7539
9	-2.8875

The following SQL statement estimates the value of the data missing from the fourth row:

```
SELECT TS_ESTIMATE_MISSING(DATA,0) OVER (ORDER BY
rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM EST_MISSING_DATASET
```

Sybase IQ returns the following nine rows, replacing the NULL value with 1.0278:

Table 4-43: Values returned from TS_ESTIMATE_MISSING

res
2.8223
-0.5721
2.2771
1.0278
1.2648
1.0278
0.6991

res
-1.7539
-2.8875

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library

TS_LACK OF FIT function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Performs the lack-of-fit test for a univariate time series or transfer function, given the appropriate correlation function.

Syntax

TS_LACK OF FIT (*timeseries_expression*, *p_value*, *q_value*, *lagmax, [tolerance]*)

OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

p_value An integer containing the number of autoregressive parameters.

q_value An integer containing the number of moving average parameters.

lagmax An integer containing the maximum lag of the correlation function.

tolerance (An optional parameter). A floating-point value level used to determine convergence of the nonlinear least-squares algorithm. The default value is 0.

window-spec TS_LACK_OF_FIT is an OLAP function requiring an OVER () clause.

Usage

This function returns a double-precision floating-point value containing the lack-of-fit statistic (q) for the time series. TS_LACK_OF_FIT calls the function imsls_d_lack_of_fit in the IMSL libraries.

IMSL mapping

The arguments of TS_LACK_OF_FIT map to the IMSL library function imsls_d_lack_of_fit as follows:

```
params = imsls_d_arma(n_objs, z[], p, q,
    IMSLS_LEAST_SQUARES,
    IMSLS_CONVERGENCE_TOLERANCE, tolerance,
    IMSL_RESIDUAL, &residual, 0);
correlations = imsls_d_autocorrelation(n_objs-
p+lagmax, residuals, lagmax, 0);
result = imsls_d_lack_of_fit(n_objs, correlations,
lagmax, npfree, 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

p Maps to the *p_value* argument defined in TS_LACK_OF_FIT.

q Maps to the *q_value* argument defined in TS_LACK_OF_FIT.

lagmax Maps to the *lagmax* argument defined in TS_LACK_OF_FIT.

npfree Derived from p + q.

tolerance An optional argument using IMSLS_CONVERGENCE_TOLERANCE. If null, the IMSL library applies a default value and does not use IMSLS_CONVERGENCE_TOLERANCE.

For detailed information on how the IMSL function imsls_d_lack_of_fit performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_LACK_OF_FIT function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-44: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381
5	2.77555
6	2.73657

rownum	data
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422
42	4.11499
43	2.77188

rownum	data
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns the lack of fit statistic on data from the data column:

```
SELECT TS_LACK_OF_FIT(data,1,1,5,0.225) OVER (ORDER BY
rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-45: Values returned from TS_LACK_OF_FIT

res
3.96751
3.96751
3.96751
3.96751
3.96751
3.96751
3.96751
3.96751
3.96751
3.96751
...
3.96751

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise.

See also

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library

TS_LACK OF FIT_P function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Performs the lack-of-fit test for a univariate time series. This function is identical to “TS_LACK OF FIT function [Time Series]” on page 297, except that it returns the p-value of q, rather than returning q.

Syntax

TS_LACK OF FIT_P (*timeseries_expression*, *p_value*, *q_value*, *lagmax,* [*tolerance]*)

OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

p_value An integer containing the number of autoregressive parameters.

q_value An integer containing the number of moving average parameters.

lagmax An integer containing the maximum log of the correlation function.

tolerance (An optional parameter). A floating-point value level used to determine convergence of the nonlinear least-squares algorithm. The default value is 0.

window-spec TS_LACK_OF_FIT_P is an OLAP function requiring an OVER () clause.

Usage

This function returns a double-precision floating-point value containing the p-value of the lack-of-fit statistic (q) for the time series. TS_LACK_OF_FIT_P calls the function imsls_d_lack_of_fit in the IMSL libraries.

IMSL mapping

The arguments of TS_LACK_OF_FIT_P map to the IMSL library function imsls_d_lack_of_fit as follows:

```
params = imsls_d_arma(n_objs, z[], p, q,
    IMSLS_LEAST_SQUARES,
    IMSLS_CONVERGENCE_TOLERANCE, tolerance,
    IMSL_RESIDUAL, &residual, 0);
correlations = imsls_d_autocorrelation(n_objs-
p+lagmax, residuals, lagmax, 0);
result = imsls_d_lack_of_fit(n_objs, correlations,
lagmax, npfree, 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

p Maps to the *p_value* argument defined in TS_LACK_OF_FIT_P.

q Maps to the *q_value* argument defined in TS_LACK_OF_FIT_P.

lagmax Maps to the *lagmax* argument defined in TS_LACK_OF_FIT_P.

npfree Derived from p + q.

tolerance An optional argument using IMSLS_CONVERGENCE_TOLERANCE. If null, the IMSL library applies a default value and does not use IMSLS_CONVERGENCE_TOLERANCE.

For detailed information on how the IMSL function imsls_d_lack_of_fit performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

Example

This example shows an input data table, a SQL statement containing the TS_LACK_OF_FIT_P function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-46: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381
5	2.77555
6	2.73657
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405

rownum	data
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422
42	4.11499
43	2.77188
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns the p value of the lack of fit statistic on data from the data column:

```
SELECT TS_LACK_OF_FIT_P(data,1,1,5,0.225) OVER (ORDER
BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-47: Values returned from TS_LACK_OF_FIT_P

res
0.735006
0.735006
0.735006
0.735006
0.735006
0.735006
0.735006
0.735006
0.735006
0.735006
...
0.735006

See also

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library

TS_MAX_ARMA_AR function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns the requested autoregressive estimate.

Syntax

TS_MAX_ARMA_AR (*timeseries_expression*, *ar_count*, *ar_elem*)

OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

ar_count An integer containing the number of autoregressive values to compute.

ar_elem An integer identifying which element in the computed autoregressive array is to be returned. The integer must be greater than 0 and less than or equal to *ar_count*.

window-spec TS_MAX_ARMA_AR is an OLAP function requiring an OVER () clause.

Usage

This function returns a double-precision floating-point value containing the autoregressive estimate. TS_MAX_ARMA_AR calls the function imsls_d_max_arma in the IMSL libraries.

IMSL mapping

The arguments of TS_MAX_ARMA_AR map to the IMSL library function imsls_d_max_arma as follows:

```
params = imsls_d_max_arma(n_objs, z[], p, q, 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

p Maps to the *ar_count* argument.

q =1.

For detailed information on how the IMSL function imsls_d_max_arma performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example 1

This example shows an input data table, a SQL statement containing the TS_MAX_ARMA_AR function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-48: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381
5	2.77555
6	2.73657

rownum	data
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422
42	4.11499
43	2.77188

rownum	data
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns the second element from an array containing two autoregressive estimates of data from the data column:

```
SELECT TS_MAX_ARMA_AR(data,2,2) OVER (ORDER BY rownum
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-49: Values returned from TS_MAX_ARMA_AR Example 1

res
0.179748
0.179748
0.179748
0.179748
0.179748
0.179748
0.179748
0.179748
0.179748
0.179748
...
0.179748

Example 2

This example provides a sample query that returns two columns of results from the DATASET table—the first and second elements of the autoregressive estimates. See Table 4-48 on page 305 for the DATASET table.

```
SELECT TS_MAX_ARMA_AR(data,2,1) OVER (ORDER BY rownum
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS ar_elem1, TS_MAX_ARMA_AR(data,2,2) OVER
(ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND
UNBOUNDED FOLLOWING) AS ar_elem2 FROM DATASET
```

Sybase IQ returns 50 rows of data, each containing the same two values:

Table 4-50: Values returned from TS_MAX_ARMA_AR Example 2

ar_elem1	ar_elem2
0.731164	0.179748
0.731164	0.179748
0.731164	0.179748
0.731164	0.179748
0.731164	0.179748
0.731164	0.179748
0.731164	0.179748
0.731164	0.179748
0.731164	0.179748
0.731164	0.179748
0.731164	0.179748
...	...
0.731164	0.179748

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library

TS_MAX_ARMA_CONST function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns the constant estimate.

Syntax

TS_MAX_ARMA_CONST (*timeseries_expression*)

OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

window-spec TS_MAX_ARMA_CONST is an OLAP function requiring an OVER () clause.

Usage

This function returns a double-precision floating-point value containing the constant estimate. TS_MAX_ARMA_CONST calls the function imsls_d_arma in the IMSL libraries.

IMSL mapping

The arguments of TS_MAX_ARMA_CONST map to the IMSL library function imsls_d_arma as follows:

```
params = imsls_d_max_arma(n_objs, z, p, q, 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

p = 1.

q = 1.

For detailed information on how the IMSL function imsls_d_arma performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_MAX_ARMA_CONST function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-51: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381
5	2.77555
6	2.73657
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859

rownum	data
15	0.833405
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422
42	4.11499
43	2.77188
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns the constant estimate of the maximum likelihood autoregressive calculation on data from the data column:

```
SELECT TS_MAX_ARMA_CONST(data) OVER (ORDER BY rownum
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-52: Values returned from TS_MAX_ARMA_CONST

res
0.107555
0.107555
0.107555
0.107555
0.107555
0.107555
0.107555
0.107555
0.107555
0.107555
...
0.107555

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library

TS_MAX_ARMA_LIKELIHOOD function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns likelihood value (ln) for the fitted model.

Syntax

TS_MAX_ARMA_LIKELIHOOD (*timeseries_expression*)

OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

window-spec TS_MAX_ARMA_LIKELIHOOD is an OLAP function requiring an OVER () clause.

Usage

This function returns a double-precision floating-point value containing the value of *-2*(ln(likelihood))*. TS_MAX_ARMA_LIKELIHOOD calls the function imsls_d_max_arma in the IMSL libraries.

IMSL mapping

The arguments of TS_MAX_ARMA_LIKELIHOOD map to the IMSL library function imsls_d_max_arma as follows:

```
params = imsls_d_max_arma(n_objs, z, p, q,
IMSLS_LOG_LIKELIHOOD, &likelihood, 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

p = 1.

q = 1.

likelihood Provided by the function call. Contains the log likelihood result.

For detailed information on how the IMSL function imsls_d_max_arma performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_MAX_ARMA_LIKELIHOOD function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-53: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381

rownum	data
5	2.77555
6	2.73657
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422

rownum	data
42	4.11499
43	2.77188
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns the likelihood value of the maximum likelihood estimation on data from the data column:

```
SELECT TS_MAX_ARMA_LIKELIHOOD(data) OVER (ORDER BY
ROWNUM rows BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-54: Values returned from TS_MAX_ARMA_LIKELIHOOD

res
-11.7818
-11.7818
-11.7818
-11.7818
-11.7818
-11.7818
-11.7818
-11.7818
-11.7818
-11.7818
...
-11.7818

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library

TS_MAX_ARMA_MA function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns the requested moving average estimate.

Syntax

TS_MAX_ARMA_MA (*timeseries_expression*, *ma_count*, *ma_elem*)
OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

ma_count An integer containing the number of auto-regressive values to compute.

ma_elem An integer specifying element in the computed moving average array to return. The integer must be greater than zero, and less than or equal to *ma_count*.

window-spec TS_MAX_ARMA_MA is an OLAP function requiring an OVER () clause.

Usage

This function returns double-precision floating-point value containing the autoregressive estimate. TS_MAX_ARMA_MA calls the function imsls_d_max_arma in the IMSL libraries.

IMSL mapping

The arguments of TS_MAX_ARMA_MA map to the IMSL library function imsls_d_max_arma as follows:

```
params = imsls_d_max_arma(n_objs, z[], p, q, 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

p =1.

q Maps to the TS_MAX_ARMA_MA argument *ma_count*.

For detailed information on how the IMSL function imsls_d_max_arma performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_MAX_ARMA_MA function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-55: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381
5	2.77555
6	2.73657
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423

rownum	data
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422
42	4.11499
43	2.77188
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns the moving average of the maximum likelihood estimation on data from the data column:

```
SELECT TS_MAX_ARMA_MA(DATA,5,4) OVER (ORDER BY ROWNUM
rows BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-56: Values returned from TS_MAX_ARMA_MA

res
-0.035006
-0.035006
-0.035006
-0.035006
-0.035006
-0.035006
-0.035006
-0.035006
-0.035006
-0.035006
...
-0.035006

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library

TS_OUTLIER_IDENTIFICATION function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Detects and determines outliers and simultaneously estimates the model parameters in a time series where the underlying outlier-free series follows a general seasonal or non-seasonal ARMA model.

Syntax

TS_OUTLIER_IDENTIFICATION (*timeseries_expression*, *p_value*, *q_value*, *s_value*, *d_value*, [, *delta_value*[, *critical_value*]])

OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

p_value An integer containing the p-portion of the autoregressive integrated moving average (ARIMA) (p, 0, q)x(0, d, $0)_s$ model that the outlier free series follows.

q_value An integer containing the q-portion of the ARIMA (p, 0, q)x(0, d, $0)_s$ model that the outlier free series follows.

s_value An integer containing the s-portion of the ARIMA (p, 0, q)x(0, d, $0)_s$ model that the outlier free series follows.

d_value An integer containing the d-portion of the ARIMA (p, 0, q)x(0, d, $0)_s$ model that the outlier free series follows.

delta_value (Optional) A double precision float value containing the dampening effect parameter used in detecting a temporary change outlier. The integer must be greater than 0 and less than 1. The default value is 0.7.

critical_value (Optional) A double-precision float value used as a threshold for outlier detection. The default is 3.0.

window-spec TS_OUTLIER_IDENTIFICATION is an OLAP function requiring an OVER () clause with an unbounded window. This function does not support value-based windows; for example, you cannot use a range specifier in the OVER () clause.

Usage

This function returns an outlier-free time series. TS_OUTLIER_IDENTIFICATION calls the function imsls_d_ts_outlier_identification in the IMSL libraries.

IMSL mapping

The arguments of TS_OUTLIER_IDENTIFICATION map to the IMSL library function imsls_d_ts_outlier_identification as follows:

```
params = imsls_d_ts_outlier_identification(n_objs,
model[], z[], 0);
```

n_objs Contains the number of rows in the current window frame.

model An array containing the TS_OUTLIER_IDENTIFICATION arguments *p_value*, *s_value*, *q_value*, *d_value*:
model[0] = p_value;
model[1] = s_value;
model[2] = q_value;
model[3] = d_value;

z[] Contains the value of *timeseries_expression* for the current window frame.

If *delta_value* is non-null, the arguments of TS_OUTLIER_IDENTIFICATION map to the IMSL library function imsls_d_ts_outlier_identification as follows:

```
params = imsls_d_ts_outlier_identification(n_objs,
model[], z[], IMSL_DELTA, delta_value, 0);
```

If *critical_value* is non-null, the arguments of TS_OUTLIER_IDENTIFICATION map to the IMSL library function imsls_d_ts_outlier_identification as follows:

```
params = imsls_d_ts_outlier_identification(n_objs,
model[], z[],IMSL_CRITICAL, critical_value, 0);
```

If both *delta_value* and *critical_value* are non-null, the arguments of TS_OUTLIER_IDENTIFICATION map to the IMSL library function imsls_d_ts_outlier_identification as follows:

```
params = imsls_d_ts_outlier_identification(n_objs,
model[], z[], IMSL_DELTA, delta_value, IMSL_CRITICAL,
critical_value, 0);
```

For detailed information on how the IMSL function imsls_d_ts_outlier_identification performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_OUTLIER_IDENTIFICATION function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-57: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381
5	2.77555
6	2.73657
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405
16	2.29075

rownum	data
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422
42	4.11499
43	2.77188
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement detects and determines outliers on data from the data column:

```
SELECT TS_OUTLIER_IDENTIFICATION(data,1,1,1,1,0.7,3.0)
OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING
AND UNBOUNDED FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows:

Table 4-58: Values returned from TS_OUTLIER_IDENTIFICATION

res
0.315523
0.485859
0.676886
1.97381
2.77555
2.73657
2.64233
4.26118
3.13641
4.16566
2.95952
2.14504
1.98799
0.805859
0.833405
2.29075
1.30045
0.467122
-0.170107
-0.256657
-0.382597
-0.505511
-1.90147
-0.981688
-1.43116
-1.39389
-2.34823
-2.91122
-0.927423
-0.044383
-0.389648

res
0.545008
0.614096
0.364668
1.16043
-0.654063
0.616094
2.00875
1.86696
2.80171
3.78422
4.11499
2.77188
4.00312
4.21298
5.00413
4.74498
4.89621
3.93273
4.31592

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library

TS_PARTIAL_AUTOCORRELATION function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

Calculates the sample partial autocorrelation function of a stationary time series.

Syntax

TS_PARTIAL_AUTOCORRELATION (*timeseries_expression*, *lagmax*, *lag_elem*)

OVER (*window-spec*)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series.

lagmax An integer containing the maximum lag of autocovariance, autocorrelations, and standard errors of autocorrelations to be calculated. The integer must be greater than or equal to 1, and less than the number of elements in the time series.

lag_elem An integer identifying the element in the autocorrelation array to return. The integer must be greater than 0 and less than or equal to *lagmax*.

window-spec TS_PARTIAL_AUTOCORRELATION is an OLAP function requiring an OVER () clause.

Usage

This function returns an outlier-free time series. TS_PARTIAL_AUTOCORRELATION calls the function imsls_d_autocorrelation and imsls_d_partial_autocorrelation in the IMSL libraries.

IMSL mapping

The arguments of TS_PARTIAL_AUTOCORRELATION map to the IMSL library functions imsls_d_autocorrelation and imsls_d_partial_autocorrelation as follows:

```
params = imsls_d_autocorrelation(n_objs, z[], lagmax,
0);
```

```
result = imsls_d_partial_autocorrelation(lagmax,
params, 0);
```

n_objs Contains the number of rows in the current window frame.

z[] Contains the value of *timeseries_expression* for the current window frame.

lagmax Maps to the TS_PARTIAL_AUTOCORRELATION argument *lagmax*.

For detailed information on how the IMSL functions imsls_d_autocorrelation and imsls_d_partial_autocorrelation perform time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_PARTIAL_AUTOCORRELATION function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

Table 4-59: Input data table DATASET

rownum	data
1	0.315523
2	0.485859
3	0.676886
4	1.97381
5	2.77555
6	2.73657
7	2.64233
8	4.26118
9	3.13641
10	4.16566
11	2.95952
12	2.14504
13	1.98799
14	0.805859
15	0.833405
16	2.29075
17	1.30045
18	0.467122
19	-0.170107
20	-0.256657
21	-0.382597
22	-0.505511
23	-1.90147
24	-0.981688
25	-1.43116
26	-1.39389
27	-2.34823
28	-2.91122
29	-0.927423
30	-0.044383
31	-0.389648
32	0.545008
33	0.614096
34	0.364668
35	1.16043
36	-0.654063

rownum	data
37	0.616094
38	2.00875
39	1.86696
40	2.80171
41	3.78422
42	4.11499
43	2.77188
44	4.00312
45	4.21298
46	5.00413
47	4.74498
48	4.89621
49	3.93273
50	4.31592

The following SQL statement returns the first element from an array containing partial autocorrelations of data from the data column:

```
SELECT TS_PARTIAL_AUTOCORRELATION(data,1,1) OVER (ORDER
BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-60: Values returned from TS_PARTIAL_AUTOCORRELATION

res
0.883453
0.883453
0.883453
0.883453
0.883453
0.883453
0.883453
0.883453
0.883453
0.883453
...
0.883453

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library

TS_VWAP function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function

VWAP stands for volume-weighted average price. TS_VWAP calculates the ratio of the value traded to the total volume traded over a particular time horizon. VWAP is a measure of the average price of a stock over a defined trading horizon. You can use TS_VWAP as both a simple and an OLAP-style aggregate function.

Unlike the other time series functions, TS_VWAP does not call the IMSL libraries.

Syntax 1

TS_VWAP (*price_expression*, *volume_expression*)

Syntax 2

TS_VWAP (*price_expression*, *volume_expression*)
OVER (*window-spec*)

Parameters

price_expression A numeric expression specifying the price to be incorporated into a volume-weighted average.

volume_expression A numeric expression specifying the volume to be used in calculating a volume-weighted average.

window-spec If used with Syntax 2, TS_VWAP is an OLAP function requiring an OVER () clause.

Usage

Sybase IQ calculates TS_VWAP using the following formula:

Figure 4-1: VWAP calculation

$$Pvwap = \frac{\Sigma j\ Pj \cdot Qj}{\Sigma j Qj}$$

Pvwap = volume weighted average price
Pj = price of trade j.
Qj = quantity of trade j.
j = an individual trade that occurred during the time horizon.

Example

This example shows an input data table, a SQL statement containing the TS_VWAP function, and the data values returned by the function. This example uses the following table (called VWAP_DATASET) as its input data. The VWAP_DATASET table contains three rows of time series data:

Table 4-61: Input data table VWAP_DATASET

rownum	price	volume
1	1	1
2	2	2
3	5	1

The following SQL statement calculates the volume weighted average price:

```
SELECT TS_VWAP(price,volume) OVER (ORDER BY rownum ROWS
BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS
res FROM VWAP_DATASET
```

Sybase IQ returns three rows:

Table 4-62: Values returned from TS_VWAP

res
2.5
2.5
2.5

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

UCASE function [String]

Function
Converts all characters in a string to uppercase.

Syntax
UCASE (*string-expression*)

Parameters
string-expression The string to be converted to uppercase.

See “REPLACE function [String]” on page 242 for more information.

Example
The following statement returns the value “CHOCOLATE”:

```
SELECT UCASE( 'ChocoLate' ) FROM iq_dummy
```

Usage
The result data type of a UCASE function is a LONG VARCHAR. If you use UCASE in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set UCASE to the correct data type and size.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** UCASE is not supported by Adaptive Server Enterprise, but UPPER provides the same feature in a compatible manner

See also
“LCASE function [String]” on page 194

“UPPER function [String]” on page 329

UPPER function [String]

Function
Converts all characters in a string to uppercase.

Syntax
UPPER (*string-expression*)

Parameters
string-expression The string to be converted to uppercase.

See “REPLACE function [String]” for more information.

Example
The following statement returns the value “CHOCOLATE”:

```
SELECT UPPER( 'ChocoLate' ) FROM iq_dummy
```

Usage
The result data type of an UPPER function is a LONG VARCHAR. If you use UPPER in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set UPPER to the correct data type and size.

Standards and compatibility

- **SQL92** This function is SQL92 compatible.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also
“LCASE function [String]” on page 194

"LOWER function [String]" on page 200

"UCASE function [String]" on page 329

USER_ID function [System]

Function

Returns an integer user identification number.

Syntax

USER_ID ([*user-name*])

Parameters

user-name The user name.

Examples

The following statement returns the user identification number 1:

```
SELECT USER_ID ('DBA') FROM iq_dummy
```

The following statement returns the user identification number 0:

```
SELECT USER_ID ('SYS') FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ

See also

"SUSER_ID function [System]" on page 266

"USER_NAME function [System]" on page 330

USER_NAME function [System]

Function

Returns the user name.

Syntax

USER_NAME ([*user-id*])

Parameters

user-id The user identification number.

Examples

The following statement returns the value "DBA":

```
SELECT USER_NAME ( 1 ) FROM iq_dummy
```

The following statement returns the value "SYS":

```
SELECT USER_NAME ( 0 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ. In Adapter Server Enterprise, USER_NAME returns the user name, not the server user name.

See also

"SUSER_NAME function [System]" on page 266

"USER_ID function [System]" on page 330

UUIDTOSTR function [String]

Function
Converts a unique identifier value (UUID, also known as GUID) to a string value.

Syntax
UUIDTOSTR (*uuid-expression*)

Parameters
uuid-expression A unique identifier value.

Example
To convert a unique identifier value into a readable format, execute a query similar to:

```
CREATE TABLE T3 (
pk uniqueidentifier primary key,c1 int);
INSERT INTO T3 (pk, c1)
values (0x12345678123456789012123456789012, 1)
SELECT UUIDTOSTR(pk) FROM T3
```

Usage
Converts a unique identifier to a string value in the format *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*, where x is a hexadecimal digit. If the binary value is not a valid unique identifier, NULL is returned.

Standards and compatibility

- **SQL92** Vendor extension
- **SQL99** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise.

See also
"NEWID function [Miscellaneous]" on page 208

"STRTOUUID function [String]" on page 263

UNIQUEIDENTIFIER in "Binary data types" on page 77

VAR_POP function [Aggregate]

Function
Computes the statistical variance of a population consisting of a numeric-expression, as a DOUBLE.

Syntax
VAR_POP ([ALL] *expression*)

Parameters
expression The expression (commonly a column name) whose population-based variance is calculated over a set of rows.

Examples
The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year( ShipDate ) AS Year, quarter( ShipDate )
   AS Quarter, AVG( Quantity ) AS Average,
   VAR_POP( Quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
   ORDER BY Year, Quarter
```

Year	Quarter	Average	Variance
2000	1	25.775148	203.9021
2000	2	27.050847	225.8109
...	...	...	...

Usage

Computes the population variance of the provided *value expression* evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of *value expression*, from the mean of *value expression*, divided by the number of rows (remaining) in the group or partition.

Population-based variances are computed according to the following formula:

$$\frac{\sum (x_i - \bar{x})^2}{n}$$

Standards and compatibility

- **SQL99** SQL/foundation feature outside of core SQL
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

“Analytical functions” on page 104

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

VAR_SAMP function [Aggregate]

Function

Computes the statistical variance of a sample consisting of a numeric-expression, as a DOUBLE.

Note VAR_SAMP is an alias of VARIANCE.

Syntax

VAR_SAMP ([ALL] *expression*)

Parameters

expression The expression (commonly a column name) whose sample-based variance is calculated over a set of rows.

Examples

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year( ShipDate ) AS Year, quarter( ShipDate )
   AS Quarter, AVG( Quantity ) AS Average,
   VAR_SAMP( Quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
   ORDER BY Year, Quarter
```

Year	Quarter	Average	Variance
2000	1	25.775148	205.1158
2000	2	27.050847	227.0939
...	...	...	...

Usage

Computes the sample variance of *value expression* evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of *value expression*, from the mean of *value expression*, divided by one less than the number of rows (remaining) in the group or partition.

NULL returns NULL for a one-element input set in IQ 12.7 and later. In versions earlier than 12.7, NULL returned zero.

Variances are computed according to the following formula, which assumes a normal distribution:

$$\frac{\sum (x_i - \bar{x})^2}{n}$$

Standards and compatibility

- **SQL99** SQL/foundation feature outside of core SQL
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"Analytical functions" on page 104

"VARIANCE function [Aggregate]" on page 333

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

VARIANCE function [Aggregate]

Function

Returns the variance of a set of numbers.

Syntax

VARIANCE ([ALL] *expression*)

Parameters

expression Any numeric data type (FLOAT, REAL, or DOUBLE) expression.

Examples

Given this data:

```
SELECT Salary FROM Employees WHERE DepartmentID = 300
```

salary
51432.000
57090.000
42300.000
43700.00
36500.000
138948.000
31200.000
58930.00
75400.00

The following statement returns the value 1063923790.99999994:

```
SELECT VARIANCE ( Salary ) FROM Employees
WHERE DepartmentID = 300
```

Given this data:

```
SELECT UnitPrice FROM Products WHERE name = 'Tee Shirt'
```

UnitPrice
9.00
14.00
14.00

The following statement returns the value 8.33333333333334327:

```
SELECT VARIANCE ( UnitPrice ) FROM Products
WHERE name = 'Tee Shirt'
```

Usage

The formula used to calculate VARIANCE is

$$var = \frac{n\sum x^2 - \left(\sum x\right)^2}{n(n-1)}$$

VARIANCE returns a result of data type double-precision floating-point. If applied to the empty set, the result is NULL, which returns NULL for a one-element input set.

VARIANCE does not support the keyword DISTINCT. A syntax error is returned if DISTINCT is used with VARIANCE.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"STDDEV function [Aggregate]" on page 257

"VAR_SAMP function [Aggregate]" on page 332

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

WEEKS function [Date and time]

Function

Returns the number of weeks since an arbitrary starting date/time, returns the number of weeks between two specified date/times, or adds the specified integer-expression number of weeks to a date/time.

Syntax

WEEKS (*datetime-expression*
| *datetime-expression, datetime-expression*
| *datetime-expression, integer-expression*)

Parameters

datetime-expression A date and time.

integer-expression The number of weeks to be added to the *datetime-expression.* If *integer-expression* is negative, the appropriate number of weeks are subtracted from the date/time value. Hours, minutes, and seconds are ignored. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a DATETIME data type.

For information on casting data types, see "CAST function [Data type conversion]" on page 133.

Examples

The following statement returns the value 104278:

```
SELECT WEEKS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 9, to signify the difference between the two dates:

```
SELECT WEEKS( '1999-07-13 06:07:12',
    '1999-09-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the timestamp value 1999-06-16 21:05:07.000:

```
SELECT WEEKS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5) FROM iq_dummy
```

Usage

Weeks are defined as going from Sunday to Saturday, as they do in a North American calendar. The number returned by the first syntax is often useful for determining if two dates are in the same week.

```
WEEKS ( invoice_sent ) = WEEKS ( payment_received ) FROM
iq_dummy
```

In the second syntax, the value of WEEKS is calculated from the number of Sundays between the two dates. Hours, minutes, and seconds are ignored. This function is not affected by the DATE_FIRST_DAY_OF_WEEK option.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

WEIGHTED_AVG function [Aggregate]

Function

Calculates an arithmetically (or linearly) weighted average. A weighted average is an average in which each quantity to be averaged is assigned a weight. Weightings determine the relative importance of each quantity that make up the average.

Syntax

WEIGHTED_AVG (*expression*)

OVER (*window-spec*)

window-spec: See the Usage section, below.

Parameters

expression A numeric expression for which a weighted value is being computed.

Usage

Use the WEIGHTED_AVG function to create a weighted moving average. In a weighted moving average, weights decrease arithmetically over time. Weights decrease from the highest weight for the most recent data points, down to zero.

Figure 4-2: WEIGHTED_AVG calculation

$$WMA_M = \frac{np_M + (n-1)p_{M-1} + \cdots + 2p_{M-n+2} + p_{M-n+1}}{n + (n-1) + \cdots + 2 + 1}$$

To exaggerate the weighting, you can average two or more weighted moving averages together, or use an EXP_WEIGHTED_AVG function instead.

You can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

window-spec:

- Must contain an ORDER BY specifier.

- Cannot contain FOLLOWING or RANGE specifiers.
- The second argument of the ROW specifier—if provided—must be CURRENT ROW.
- Cannot contain NULL values.
- Cannot contain the DISTINCT specifier.
- UNBOUNDED PRECEDING is supported, but may result in poor performance if used

For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example returns a weighted average of salaries by department for employees in Florida, with the salary of recently hired employees contributing the most weight to the average:

```
SELECT DepartmentID, Surname, Salary,
WEIGHTED_AVG(Salary) OVER (PARTITION BY DepartmentID
ORDER BY YEAR(StartDate) DESC) as "W_AVG"
FROM Employees
WHERE State IN ('FL') ORDER BY DepartmentID
```

The returned result set is:

Table 4-63: WEIGHTED_AVG result set

DepartmentID	Surname	Salary	W_AVG
100	Lull	87900.000	87900.000000
100	Gowda	59840.000	69193.333333
200	Sterling	64900.000	64900.000000
200	Kelly	87500.000	79966.666667
300	Litton	58930.000	58930.000000
400	Evans	68940.000	68940.000000
400	Charlton	28300.000	41846.666667
400	Francis	53870.000	47858.333333

Standards and compatibility

- **SQL2008** Vendor extension

WIDTH_BUCKET function [Numerical]

Function

For a given expression, the WIDTH_BUCKET function returns the bucket number that the result of this expression will be assigned after it is evaluated.

Syntax

WIDTH_BUCKET (*expression*, *min_value*, *max_value*, *num_buckets*)

Parameters

expression is the expression for which the histogram is being created. This expression must evaluate to a numeric or datetime value or to a value that can be implicitly converted to a numeric or datetime value. If *expr* evaluates to null, then the expression returns null.

min_value An expression that resolves to the end points of the acceptable range for *expr*. Must also evaluate to numeric or datetime values and cannot evaluate to null.

max_value An expression that resolves to the end points of the acceptable range for *expr*. Must also evaluate to numeric or datetime values and cannot evaluate to null.

num_buckets Is an expression that resolves to a constant indicating the number of buckets. This expression must evaluate to a positive integer.

Examples

The following example creates a ten-bucket histogram on the credit_limit column for customers in Massachusetts in the sample table and returns the bucket number ("Credit Group") for each customer. Customers with credit limits greater than the maximum value are assigned to the overflow bucket, 11:

```
select EmployeeID, Surname, Salary,
WIDTH_BUCKET(Salary, 29000, 60000, 4) "Wages" from
Employees where State = 'FL' order by "Wages"
```

```
EMPLOYEEID   SURNAME              SALARY         Wages
----------   -------              ------         -----
888          Charlton          28300.000             0
1390         Litton            58930.000             4
207          Francis           53870.000             4
266          Gowda             59840.000             4
445          Lull              87900.000             5
1021         Sterling          64900.000             5
902          Kelly             87500.000             5
1576         Evans             68940.000             5
```

When the bounds are reversed, the buckets are open-closed intervals. For example: WIDTH_BUCKET (*credit_limit*, *5000*, *0*, *5*). In this example, bucket number 1 is (4000, 5000], bucket number 2 is (3000, 4000], and bucket number 5 is (0, 1000]. The overflow bucket is numbered 0 (5000, +infinity), and the underflow bucket is numbered 6 (-infinity, 0].

Usage

You can generate equiwidth histograms with the WIDTH_BUCKET function. Equiwidth histograms divide data sets into buckets whose interval size (highest value to lowest value) is equal. The number of rows held by each bucket will vary. A related function, NTILE, creates equiheight buckets.

Equiwidth histograms can be generated only for numeric, date or datetime data types; therefore, the first three parameters should be all numeric expressions or all date expressions. Other types of expressions are not allowed. If the first parameter is NULL, the result is NULL. If the second or the third parameter is NULL, an error message is returned, as a NULL value cannot denote any end point (or any point) for a range in a date or numeric value dimension. The last parameter (number of buckets) should be a numeric expression that evaluates to a positive integer value; 0, NULL, or a negative value will result in an error.

Buckets are numbered from 0 to (n+1). Bucket 0 holds the count of values less than the minimum. Bucket(n+1) holds the count of values greater than or equal to the maximum specified value.

Standards and compatibility

- **SQL03** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"NTILE function [Analytical]" on page 212, which creates equiheight histograms.

Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*

YEAR function [Date and time]

Function

Returns a 4-digit number corresponding to the year of the given date/time.

Syntax

YEAR (*datetime-expression*)

Parameters

datetime-expression A date and time.

Example

The following statement returns the value 1998:

```
SELECT YEAR( '1998-07-13 06:07:12' ) FROM iq_dummy
```

Usage

The YEAR function is the same as the first syntax of the YEARS function.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"YEARS function [Date and time]" on page 339

YEARS function [Date and time]

Function

Returns a 4-digit number corresponding to the year of a given date/time, returns the number of years between two specified date/times, or adds the specified integer-expression number of years to a date/time.

Syntax

YEARS (*datetime-expression*
| *datetime-expression, datetime-expression*
| *datetime-expression, integer-expression*)

Parameters

datetime-expression A date and time.

integer-expression The number of years to be added to the *datetime-expression.* If *integer-expression* is negative, the appropriate number of years are subtracted from the datetime value. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a DATETIME data type.

For information on casting data types, see "CAST function [Data type conversion]" on page 133.

Examples

The following statement returns the value 1998:

```
SELECT YEARS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 2, to signify the difference between the two dates.

```
SELECT YEARS( '1997-07-13 06:07:12',
   '1999-09-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the YEARS(cast('1999-05-12 21:05:07' as timestamp), 5) value 2004-05-12 21:05:07.000:

```
SELECT YEARS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5) FROM iq_dummy
```

Usage

The first syntax of the YEARS function is the same as the YEAR function.

The second syntax returns the number of years from the first date to the second date, calculated from the number of first days of the year between the two dates. The number might be negative. Hours, minutes, and seconds are ignored. For example, the following statement returns 2, which is the number of first days of the year between the specified dates:

```
SELECT YEARS ( '2000-02-24', '2002-02-24' ) FROM
iq_dummy
```

The next statement also returns 2, even though the difference between the specified dates is not two full calendar years. The value 2 is the number of first days of the year (in this case January 01, 2001 and January 01, 2002) between the two dates.

```
SELECT YEARS ( '2000-02-24', '2002-02-20' ) FROM
iq_dummy
```

The third syntax adds an *integer-expression* number of years to the given date. If the new date is past the end of the month (such as SELECT YEARS (CAST ('1992-02-29' AS TIMESTAMP), 1)), the result is set to the last day of the month. If *integer-expression* is negative, the appropriate number of years is subtracted from the date. Hours, minutes, and seconds are ignored.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

"YEAR function [Date and time]" on page 339

YMD function [Date and time]

Function

Returns a date value corresponding to the given year, month, and day of the month.

Syntax

YMD (*integer-expression1*, *integer-expression2*, *integer-expression3*)

Parameters

integer-expression1 The year.

integer-expression2 The number of the month. If the month is outside the range 1–12, the year is adjusted accordingly.

integer-expression3 The day number. The day is allowed to be any integer, the date is adjusted accordingly.

Examples

The following statement returns the value 1998-06-12:

```
SELECT YMD( 1998, 06, 12 ) FROM iq_dummy
```

If the values are outside their normal range, the date adjusts accordingly. For example, the following statement returns the value 1993-03-01:

```
SELECT YMD( 1992, 15, 1 ) FROM iq_dummy
```

The following statement returns the value 1993-02-28:

```
SELECT YMD ( 1992, 15, 1-1 ) FROM iq_dummy
```

The following statement returns the value 1992-02-29:

```
SELECT YMD ( 1992, 3, 1-1 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

CHAPTER 5

Differences from Other SQL Dialects

About this chapter

Sybase IQ conforms to the ANSI SQL89 standard, but has many additional features that are defined in the IBM DB2 and SAA specifications, as well as in the ANSI SQL92 standard.

This chapter describes those features of Sybase IQ that are not commonly found in other SQL implementations.

Contents

Sybase IQ features

The following Sybase IQ features are not found in many other SQL implementations.

Dates

Sybase IQ has date, time, and timestamp types that include year, month, day, hour, minutes, seconds, and fraction of a second. For insertions or updates to date fields, or comparisons with date fields, a free-format date is supported.

In addition, the following operations are allowed on dates:

- **date + integer** Add the specified number of days to a date.
- **date - integer** Subtract the specified number of days from a date.
- **date - date** Compute the number of days between two dates.
- **date + time** Make a timestamp out of a date and time.

Also, many functions are provided for manipulating dates and times. See Chapter 4, "SQL Functions" for a description of these.

Integrity

Sybase IQ supports both entity and referential integrity. This has been implemented via the following two extensions to the CREATE TABLE and ALTER TABLE commands.

```
PRIMARY KEY ( column-name, ... )
[NOT NULL] FOREIGN KEY [role-name]
              [(column-name, ...)]
           REFERENCES table-name [(column-name, ...)]
              [ CHECK ON COMMIT ]
```

The PRIMARY KEY clause declares the primary key for the relation. Adaptive Server IQ will then enforce the uniqueness of the primary key, and ensure that no column in the primary key contains the NULL value.

The FOREIGN KEY clause defines a relationship between this table and another table. This relationship is represented by a column (or columns) in this table which must contain values in the primary key of another table. The system then ensures referential integrity for these columns; whenever these columns are modified or a row is inserted into this table, these columns are checked to ensure that either one or more is NULL or the values match the corresponding columns for some row in the primary key of the other table. For more information, see CREATE TABLE statement.

Joins

Sybase IQ allows automatic joins between tables. In addition to the NATURAL and OUTER join operators supported in other implementations, Sybase IQ allows KEY joins between tables based on foreign-key relationships. This reduces the complexity of the WHERE clause when performing joins.

Updates

Sybase IQ allows more than one table to be referenced by the UPDATE command. Views defined on more than one table can also be updated. Many SQL implementations do not allow updates on joined tables.

Altering tables

The ALTER TABLE command has been extended. In addition to changes for entity and referential integrity, the following types of alterations are allowed:

```
ADD column data-type
MODIFY column data-type
DELETE column
RENAME new-table-name
RENAME old-column TO new-column
```

You can use MODIFY to change the maximum length of a character column, as well as converting from one data type to another. See "ALTER TABLE statement," in "SQL Statements,"in *Reference: Statements and Options*.

Subqueries not always allowed

Unlike SQL Anywhere, Sybase IQ does not allow subqueries to appear wherever expressions are allowed. Sybase IQ supports subqueries only as allowed in the SQL-1989 grammar, plus in the SELECT list of the top level query block or in the SET clause of an UPDATE statement. Sybase IQ does not support SQL Anywhere extensions.

Many SQL implementations allow subqueries only on the right side of a comparison operator. For example, the following command is valid in Sybase IQ but not valid in most other SQL implementations.

```
SELECT SurName,
          BirthDate,
          (   SELECT DepartmentName
              FROM Departments
              WHERE DepartmentID = Employees.EmployeeID
              AND DepartmentID = 200 )
FROM Employees
```

Additional functions

Sybase IQ supports several functions not in the ANSI SQL definition. See Chapter 4, "SQL Functions" for a full list of available functions.

Cursors

When using Embedded SQL, cursor positions can be moved arbitrarily on the FETCH statement. Cursors can be moved forward or backward relative to the current position or a given number of records from the beginning or end of the cursor.

CHAPTER 6

Physical Limitations

About this chapter

This chapter describes the limitations on size and number of objects in Sybase IQ databases. For limitations that apply to only one platform, see the platform-specific documentation.

Contents

Size and number limitations

Table 6-1 lists the limitations on size and number of objects in a Sybase IQ database. In most cases, computer memory and disk drive are more limiting factors.

Table 6-1: Sybase IQ database object size and number limitations

Item	Limitation
Catalog file size	Maximum is 1TB for all platforms except for Windows systems with FAT 32-file systems, which have a 4GB limit. Windows systems with NTFS support the 1TB maximum. Sybase IQ does not support creating dbspaces on NAS (Network Attached Storage) devices.
Database name size	250 bytes.
Database size	Maximum database size approximates the number of files times the file size on a particular platform, depending on the maximum disk configuration. Refer to your operating system documentation for kernel parameters that affect the maximum number of files.
Dbspace size	Raw: No limit – as large as the device allows. Operating system files: 4TB.
Field size	255 bytes for BINARY, 32,767 bytes for VARBINARY 32,767 for CHAR, VARCHAR. Up to 512 TB for 128 KB pages or 1 PB for 512 KB pages for LONG BINARY, LONG VARCHAR.
IQ page size	Must be between 64KB and 512KB.
Maximum key size	255 bytes for single-column index. 5300 bytes for multicolumn index.
Maximum length of SQL statement	The maximum length of SQL statements is limited to the amount of memory available for the IQ catalog, and to the size of the catalog stack. If your SQL statements are long, increase the catalog stack size using -gss, and increase catalog memory cache size using -c or a combination of -ch and -cl. When printing the SQL statement in error messages, the text is limited to the IQ catalog page size. To print long commands, you can start the server with an increased -gp setting, although in general Sybase recommends that you use the default of -gp 4096.
Maximum length of variable-length FILLER column	512 bytes.
Maximum number of users (connected and concurrent)	1000 on 64-bit platforms AIX, HP, Linux, and Sun Solaris. 200 on 32- and 64-bit platforms on Windows.

Item	**Limitation**
Maximum size of temp extract file	Set by TEMP_EXTRACT_SIZEn option. Platform limits are: AIX & HP-UX: 0 – 64GB Sun Solaris: 0 – 512GB Windows: 0 – 128GB Linux: 0 – 512GB
Number of columns per table	Sybase IQ supports up to 45,000 columns in a table. You may see performance degradation if you have more than 10,000 columns in a table.
Number of events per database	2^31 – 1 = 2 147 483 647.
Number of files per database	Operating system limit that user can adjust; for example, using NOFILE. Typically, 2047 files per database.
Number of indexes	2^32 (~4,000,000) per table.
Number of rows per table	Limited by table size, upper limit 2^48.
Number of stored procedures per database	2^32 – 1 = 4 294 967 295.
Number of tables or views in a single FROM clause	16 – 64, depending on the query, with join optimizer turned on.
Number of tables or views referenced per query	512.
Number of tables per database	4,293,918,719.
Number of tables per join index (number of tables that can be joined in one query block)	32.
Number of tables referenced per transaction	No limit.
Number of UNION branches per query	512. If each branch has multiple tables in the FROM clause, the limit on tables per query reduces the number of UNION branches allowed.
Number of values in an IN list	250,000.
Row size	Sybase recommends a limit of half the page size.
Table size	Limited by database size.

CHAPTER 7 System Procedures

About this chapter

This chapter documents the system-supplied stored procedures in Sybase IQ databases that you can use to retrieve system information.

Contents

System procedure overview

Sybase IQ includes the following kinds of system procedures:

- System functions that are implemented as stored procedures.
- Catalog stored procedures, for displaying system information in tabular form.
- Multiplex stored procedures, which include both of the above types of procedures, for multiplex server operations. See *"System procedures"* in *Appendix A, "Multiplex Reference,"* in *Using Sybase IQ Multiplex*.
- Transact-SQL system and catalog procedures. For a list of these system procedures, see "Adaptive Server Enterprise system and catalog procedures" on page 522.

This chapter describes system procedures.

System stored procedures related specifically to Large Object data, including sp_iqsetcompression and sp_iqshowcompression, are described in Chapter 5, "Stored Procedure Support" in *Large Objects Management in Sybase IQ*.

Syntax rules for stored procedures

Use of parentheses and quotes in stored procedure calls varies, depending on whether you enter the procedure name directly, as you can in Interactive SQL, or invoke it with a CALL statement. Some variations are permitted because the product supports both Sybase IQ SQL and Transact-SQL syntax. If you need Transact-SQL compatibility, be sure to use Transact-SQL syntax.

See Table 7-1 for an explanation of syntax variations.

Table 7-1: Stored procedure syntax variations

Syntax	Syntax type	Explanation
procedure_name ('*param*')	Sybase IQ	Quotes are required if you enclose parameters in parentheses.
procedure_name '*param*'	Sybase IQ	Parentheses are optional if you enclose parameters in quotes.
procedure_name param	Transact-SQL	If you omit quotes around parameters, you must also omit parentheses.
procedure_name	Sybase IQ or Transact-SQL	Use this syntax if you run a procedure with no parameters directly in DBISQL, and the procedure has no parameters.

Syntax	Syntax type	Explanation
call *procedure_name* (*param*='*value*')	Sybase IQ	Use this syntax to call a procedure that passes a parameter value.

When you use Transact-SQL stored procedures, you must use the Transact-SQL syntax.

Understanding statistics reported by stored procedures

Many stored procedures report information on the state of Sybase IQ at the time the procedure executes. This means that you get a snapshot view. For example, a report column that lists space in use by a connection shows only the space in use at the instant the procedure executes, not the maximum space used by that connection.

To monitor Sybase IQ usage over an extended period, use the Sybase IQ monitor, which collects and reports statistics from the time you start the monitor until you stop it, at an interval you specify.

System stored procedures

System stored procedures are owned by the user ID dbo. The system procedures in this section carry out System Administrator tasks in the IQ Store.

Note By default, the maximum length of column values displayed by DBISQLC is 30 characters. This might be inadequate for displaying output of stored procedures such as sp_iqstatus. To avoid truncated output, increase the length by selecting Command > Options from the dbisql menu select and enter a higher value for Limit Display Columns, Limit Output Columns, or both.

---

sa_dependent_views procedure

Function

Returns the list of all dependent views for a given table or view.

See "sa_dependent_views system procedure" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > System procedures > Alphabetical list of system procedures*.

sa_verify_password procedure

Function

Validates the password of the current user.

See "sa_verify_password system procedure" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > System procedures > Alphabetical list of system procedures*.

sa_get_user_status system procedure

Function

Allows you to determine the current status of users.

See "sa_get_user_status system procedure" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > System procedures > Alphabetical list of system procedures*.

sp_expireallpasswords procedure

Function

Causes all user passwords to expire immediately.

Syntax1

call sp_expireallpasswords

Syntax2

sp_expireallpasswords

See also

"sp_iqpassword procedure" on page 441

CREATE USER Statement in Chapter 1, "SQL Statements," in *Reference: Statements and Options*.

Permissions

DBA authority required.

Examples

Causes all user passwords to expire immediately:

```
call sp_expireallpasswords
```

sp_iqaddlogin procedure

Function

Adds a new Sybase IQ user account to the specified login policy.

Syntax1

call sp_iqaddlogin (‘*username_in*’, ‘*pwd*’,
[’*password_expiry_on_next_login* ’] [, ’*policy_name* ’])

Syntax2

sp_iqaddlogin ‘*username_in*’, ‘*pwd*’,
[’*password_expiry_on_next_login* ’] [, ’*policy_name* ’]

Syntax3

sp_iqaddlogin *username_in*, *pwd*, [*password_expiry_on_next_login*] [, *policy_name*]

Usage

username_in The user’s login name. Login names must conform to the rules for identifiers.

pwd The user’s password. Passwords must conform to rules for passwords, that is, they must be valid identifiers.

password_expiry_on_next_login (Optional) Specifies whether user’s password expires as soon as this user’s login is created. Default setting is OFF (password does not expire).

policy_name (Optional) Creates the user under the named login policy. If unspecified, user is created under the root login policy.

A *username_in*/*pwd* created using sp_iqaddlogin and set to expire in one day is valid all day tomorrow and invalid on the following day. In other words, a login created today and set to expire in *n* days are not usable once the date changes to the *(n+1)*th day.

Permissions

Requires DBA authority.

See also

“sp_iqpassword procedure” on page 441

CREATE USER Statement in Chapter 1, “SQL Statements,” in *Reference: Statements and Options*.

Description

Adds a new Sybase IQ user account, assigns a login policy to the user and adds the user to the ISYSUSER system table. If the user already has a user ID for the database but is not in ISYSUSER, (for example, if the user ID was added using the GRANT CONNECT statement or Sybase Central), sp_iqaddlogin adds the user to the table.

If you do not specify a login policy name when calling the procedure, Sybase IQ assigns the user to the root login policy.

Note If the maximum number of logins for a login policy is unlimited, then a user belonging to that login policy can have an unlimited number of connections.

The first user login forces a password change and assigns a login policy to the newly created user. Sybase recommends that you use CREATE USER to create new users, although, for backward compatibility, sp_iqaddlogin is still supported.

Examples

These calls add the user rose with a password irk324 under the login policy named expired_password. This example assumes the expired_password login policy already exists.

```
call sp_iqaddlogin('rose', 'irk324', 'ON',
'expired_password')
```

```
sp_iqaddlogin 'rose','irk324', 'ON', 'expired_password'
```

sp_iqbackupdetails procedure

Function

Shows all the dbfiles included in a particular backup.

Syntax

sp_iqbackupdetails *backup_id*

Parameters

backup_id Specifies the backup operation transaction identifier.

Note You can obtain the *backup_id* value from the SYSIQBACKUPHISTORY table. Run the following query:
select * from sysiqbackuphistory

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

sp_iqbackupdetails returns the following:

Table 7-2: sp_iqbackupdetails columns

Column name	Description
backup_id	Identifier for the backup transaction.
backup_time	Time of the backup.

Column name	Description
backup_type	Type of backup: “Full,” “Incremental since incremental,” or “Incremental since full.”
selective_type	Subtype of backup: "All inclusive", "All RW files in RW dbspaces", "Set of RO dbspace/file."
depends_on_id	Identifier for previous backup that the backup depends on.
dbspace_id	Identifier for the dbspace being backed up.
dbspace_name	Name of the dbspace from SYSIQBACKUPHISTORYDETAIL. If dbspace name matches the dbspace name in SYSDBSPACE for a given dbspace_id. Otherwise “null.”
dbspace_rwstatus	“ReadWrite” or “Read Only.”
dbspace_createid	Dbspace creation transaction identifier.
dbspace_alterid	Alter DBSPACE read-write mode transaction identifier.
dbspace_online	Status. Values are “Online” or “Offline.”
dbspace_size	Size of dbspace, in KB, at time of backup.
dbspace_backup_size	Size of data, in KB, backed up in the dbspace.
dbfile_id	Identifier for the dbfile being backed up
dbfile_name	The logical file name, if it was not renamed after the backup operation. If renamed, “null.”
dbfile_rwstatus	“ReadWrite” or “Read Only”.
dbfile_createid	Dbfile creation transaction identifier.
dbfile_alterid	Alter DBSPACE alter FILE read-write mode transaction identifier
dbfile_size in MB	Size of the dbfile, in KB.
dbfile_backup_size	Size of the dbfile backup, in KB.
dbfile_path	The dbfile path from SYSBACKUPDETAIL, if it matches the physical file path (“file_name”) in SYSDBFILE for a given dbspace_id and the dbfile_id. Otherwise “null.”

Example

Sample output from sp_iqbackupdetails:

```
backup_id   backup_time             backup_type   selective_type   depends_on_id
      883   2008-09-23 13:58:49.0   Full          All inclusive                0

dbspace_id   dbspace_name   dbspace_rwstatus   dbspace_createid
         0   system         ReadWrite                         0

dbspace_alterid   dbspace_online dbspace_size dbspace_backup_size dbfile_id
              0                0         2884                2884         0

dbfile_name dbfile_rwstatus dbfile_createid dbfile_alterid dbfile_size
     system        ReadWrite                0              0       2884
dbfile_backup_size dbfile_path
             2884  C:\\Documents and Settings\\All Users\\SybaseIQ\\demo\\iqdemo.db
```

See also "SYSIQBACKUPHISTORY system view" on page 546

sp_iqbackupsummary procedure

Function Summarizes backup operations performed.

Syntax **sp_iqbackupsummary** [*timestamp or backup_id*]

Parameters ***timestamp or backup_id*** Specifies the interval for which to report backup operations. If you specify a timestamp or a backup ID, only those records with backup_time greater than or equal to the time you enter are returned. If you specify no timestamp, the procedure returns all the backup records in ISYSIQBACKUPHISTORY.

Permissions DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description sp_iqbackupsummary returns the following:

Table 7-3: sp_iqbackupsummary columns

Column name	Description
backup_id	Identifier for the backup transaction
backup_time	Time of the backup
backup_type	Type of backup: "Full," "Incremental since incremental," or "Incremental since full"
selective_type	Subtype of backup: "All Inclusive", "All RW files in RW dbspaces", "Set of RO dbspace/file"
virtual_type	Type of virtual backup: "Non-virtual," "Decoupled," or "Encapsulated"
depends_on_id	Identifier for backup that the backup depends on
creator	Creator of the backup
backup_size	Size, in KB, of the backup
user_comment	User comment
backup_command	The backup statement issued (minus the comment)

Example Sample output of sp_iqbackupsummary:

```
backup_id   backup_time             backup_type   selective_type   virtual_type
      883   2008-09-23 13:58:49.0   Full          All inclusive    Non virtual
```

```
depends_on_id   creator   backup_size   user_comment   backup_command
            0   DBA             10864                  backup database to
                                                         'c:\\\\temp\\\\b1'
```

sp_iqcheckdb procedure

Function

Checks validity of the current database. Optionally corrects allocation problems for dbspaces or databases.

sp_iqcheckdb reads all storage in the database. On successful completion, the database free list (an internal allocation map) is updated to reflect the true storage allocation for the database. sp_iqcheckdb then generates a report listing the actions it has performed.

If an error is found, sp_iqcheckdb reports the name of the object and the type of error. sp_iqcheckdb does not update the free list if errors are detected.

sp_iqcheckdb also allows you to check the consistency of a specified table, index, index type, or the entire database.

Note sp_iqcheckdb is the user interface to the IQ database consistency checker (DBCC) and is sometimes referred to as DBCC.

Syntax

sp_iqcheckdb '*mode target* […] [resources *resource-percent*]'

This is the general syntax of sp_iqcheckdb. There are three modes for checking database consistency, and one for resetting allocation maps. The syntax for each mode is listed separately below. If mode and target are not both specified in the parameter string, Sybase IQ returns the error message:

```
At least one mode and target must be specified to DBCC.
```

Parameters

mode:
{ allocation | check | verify } | dropleaks

target:
[indextype *index-type* […]]
database | database resetclocks |
{ [indextype *index-type*] […] table *table-name*
[partition *partition-name*] […] |
index *index-name* | […]
dbspace *dbspace-name*}

Allocation mode

sp_iqcheckdb 'allocation *target* [resources *resource-percent*]'

Check mode

sp_iqcheckdb 'check *target* [resources *resource-percent*]'

Verify mode

sp_iqcheckdb 'verify *target* [resources *resource-percent*]'

Dropleaks mode

sp_iqcheckdb 'dropleaks *target* [resources *resource-percent*]'

Usage

database If the target is a database, all dbspaces must be online.

index-type One of the following index types: FP, CMP, LF, HG, HNG, WD, DATE, TIME, DTTM.

If the specified *index-type* does not exist in the target, an error message is returned. If multiple index types are specified and the target contains only some of these index types, the existing index types are processed by sp_iqcheckdb.

index-name May contain owner and table qualifiers: `[[owner.]table-name.]index-name`

If *owner* is not specified, current user and database owner (dbo) are substituted in that order. If *table* is not specified, *index-name* must be unique.

table-name May contain an owner qualifier: `[owner.]table-name`

If *owner* is not specified, current user and database owner (dbo) are substituted in that order. *table-name* cannot be a temporary or pre-join table.

Note If either the table name or the index name contains spaces, enclose the *table-name* or *index-name* parameter in double quotation marks:

```
sp_iqcheckdb 'check index "dbo.sstab.i2" resources 75'
```

---

partition-name The *partition-name* parameter contains no qualifiers. If it contains spaces, enclose it in double quotation marks.

dbspace-name The *dbspace-name* parameter contains no qualifiers. If it contains spaces, enclose it in double quotation marks.

The partition filter causes sp_iqcheckdb to examine a subset of the corresponding table's rows that belong to that partition. A partition filter on a table and table target without the partition filter are semantically equivalent when the table has only one partition.

The dbspace target examines a subset of the database's pages that belong to that dbspace. The dbspace must be online. The dbspace and database target are semantically equivalent when the table has only one dbspace.

resource-percent The input parameter *resource-percent* must be an integer greater than zero. The resources percentage allows you to limit the CPU utilization of the database consistency checker by controlling the number of threads with respect to the number of CPUs. If *resource-percent* = 100 (the default value), then one thread is created per CPU. If *resource-percent* > 100, then there are more threads than CPUs, which might increase performance for some machine configurations. The minimum number of threads is one.

Note The sp_iqcheckdb parameter string must be enclosed in single quotes and cannot be greater than 255 bytes in length.

Allocation problems can be repaired in dropleaks mode.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

sp_iqcheckdb checks the allocation of every block in the database and saves the information in the current session until the next sp_iqdbstatistics procedure is issued. sp_iqdbstatistics displays the latest result from the most recent execution of sp_iqcheckdb.

sp_iqcheckdb can perform several different functions, depending on the parameters specified. The modes for checking and repairing database consistency are:

Allocation mode Checks allocation with blockmap information for the entire database, a specific index, a specific index type, a specific partition, specific table, or a specific dbspace. Does not check index consistency.

Detects duplicate blocks (blocks for which two or more objects claim ownership) or extra blocks (unallocated blocks owned by an object).

Detects leaked blocks (allocated blocks unclaimed by any object in the specified target) for database or dbspace targets.

When the target is a partitioned table, allocation mode:

- Checks metadata of all the table’s partition allocation bitmaps
- Checks metadata of the tables allocation bitmap
- Verifies that blockmap entries are consistent with the table’s allocation bitmap
- Verifies that none of the table’s partition allocation bitmaps overlap
- Checks that rows defined in the table’s partition allocation bitmaps form a superset of the table’s existence bitmap

- Checks that rows defined in the table's partition allocation bitmaps form a superset of the table's allocation bitmap

---

Note sp_iqcheckdb cannot check all allocation problems if you specify the name of a single index, index type, or table in the input parameter string.

Run in allocation mode:

- To detect duplicate or unowned blocks (use database or specific tables or indexes as the target)
- If you encounter page header errors

The DBCC option resetclocks is used only with allocation mode. resetclocks is used with forced recovery to convert a multiplex secondary server to a coordinator. For information on multiplex capability, see *Using Sybase IQ Multiplex*. resetclocks corrects the values of internal database versioning clocks, in the event that these clocks are behind. Do not use the resetclocks option for any other purpose, unless you contact Sybase IQ Technical Support.

The resetclocks option must be run in single-user mode and is allowed only with the DBCC statement allocation database. The syntax of the resetclocks command is:

```
sp_iqcheckdb 'allocation database resetclocks'
```

Check mode Verifies that all database pages can be read for the entire database, specific index, specific index type, specific table, specific partition, or specific dbspace. If the table is partitioned, then check mode will check the table's partition allocation bitmaps.

Run in check mode if metadata, null count, or distinct count errors are returned when running a query.

Verify mode Verifies the contents of non-FP indexes with their corresponding FP indexes for the entire database, a specific index, a specific index type, specific table, specific partition, or specific dbspace. If the specified target contains all data pages for the FP and corresponding non-FP indexes, then verify mode detects the following inconsistencies:

- Missing key – a key that exists in the FP but not in the non-FP index.
- Extra key – a key that exists in the non-FP index but not in the FP index.
- Missing row – a row that exists in the FP but not in the non-FP index.
- Extra row – a row that exists in the non-FP index but not in the FP index
.

If the specified target contains only a subset of the FP pages, then verify mode can detect only the following inconsistencies:

- Missing key
- Missing row

If the target is a partitioned table, then verify mode also verifies that each row in the table or table partition has been assigned to the correct partition.

Run in verify mode if metadata, null count, or distinct count errors are returned when running a query.

Note sp_iqcheckdb does not check referential integrity or repair referential integrity violations.

Dropleaks mode When the Sybase IQ server runs in single-node mode, you can use dropleaks mode with either a database or dbspace target to reset the allocation map for the entire database or specified dbspace targets. If the target is a dbspace, then the dropleaks operation must also prevent read-write operations on the named dbspace. All dbspaces in the database or dbspace list must be online.

For information on running dropleaks mode on a multiplex, see *Using Sybase IQ Multiplex*.

The following examples illustrate the use of the sp_iqcheckdb procedure.

Example 1

In this example, sp_iqcheckdb checks the allocation for the entire database:

```
sp_iqcheckdb 'allocation database'
```

Example 2

In the second example, sp_iqcheckdb performs a detailed check on indexes i1, i2, and dbo.t1.i3. If you do not specify a new mode, sp_iqcheckdb applies the same mode to the remaining targets, as shown in the following command:

```
sp_iqcheckdb 'verify index i1 index i2 index dbo.t1.i3'
```

Example 3

You can combine all modes and run multiple checks on a database in a single session. In the following example, sp_iqcheckdb performs a quick check of partition p1 in table t2, a detailed check of index i1, and allocation checking for the entire database using half of the CPUs:

```
sp_iqcheckdb 'check table t2 partition p1 verify index
i1
allocation database resources 50'
```

Example 4

This example checks all indexes of the type FP in the database:

```
sp_iqcheckdb 'check indextype FP database'
```

Example 5

The following example verifies the FP and HG indexes in the table t1 and the LF indexes in the table t2:

```
sp_iqcheckdb 'verify indextype FP indextype HG table t1
indextype LF table t2'
```

Example 6

The following example illustrates one of the three “LVC cells” messages in the output of sp_iqcheckdb:

```
sp_iqcheckdb 'check index
EFG2JKL.ASIQ_IDX_T208_C504_FP'
------------------------------------
Index Statistics:
** Inconsistent Index:
abcd.EFG2JKL.ASIQ_IDX_T208_C504_FP ****** FP
Indexes Checked: 1
** Unowned LVC Cells: 212 ******
```

The sp_iqcheckdb LVC cells messages include:

- Unowned LVC cells
- Duplicate LVC cell rows
- Unallocated LVC cell rows

These messages indicate inconsistencies with a VARCHAR, VARBINARY, LONG BINARY (BLOB), or LONG VARCHAR (CLOB) column. Unowned LVC cells represent a small amount of unusable disk space and can safely be ignored. Duplicate and Unallocated LVC cells are serious errors that can be resolved only by dropping the damaged columns.

To drop a damaged column, create a new column from a copy of the old column, then drop the original column and rename the new column to the old column.

Note LVC is a VARCHAR or VARBINARY column with a width greater than 255. LONG BINARY (BLOB) and LONG VARCHAR (CLOB) also use LVC.

DBCC performance

The execution time of DBCC varies, depending on the size of the database for an entire database check, the number of tables or indexes specified, and the size of the machine. Checking only a subset of the database (that is, only specified tables, indexes, or index types) requires less time than checking an entire database.

The processing time of sp_iqcheckdb dropleaks mode depends on the number of dbspace targets.

Table 7-4 summarizes the actions and output of the four sp_iqcheckdb modes.

Table 7-4: Actions and output of sp_iqcheckdb modes

Mode	Errors detected	Output	Speed
Allocation	Allocation errors	Allocation statistics only	4TB per hour
Check	Allocation errors Most index errors	All available statistics	60GB per hour
Verify	Allocation errors All index errors	All available statistics	15GB per hour
Dropleaks	Allocation errors	Allocation statistics only	4TB per hour

Output

Depending on the execution mode, sp_iqcheckdb output includes summary results, errors, informational statistics, and repair statistics. The output may contain as many as three results sets, if you specify multiple modes in a single session. Error statistics are indicated by asterisks (*****), and appear only if errors are detected.

The output of sp_iqcheckdb is also copied to the Sybase IQ message file *.iqmsg*. If the DBCC_LOG_PROGRESS option is ON, sp_iqcheckdb sends progress messages to the IQ message file, allowing the user to follow the progress of the DBCC operation as it executes.

Output example

The following is an example of the output you see when you run sp_iqcheckdb 'allocation database' and there is leaked space. Leaked space is a block that is allocated according to the database free list (an internal allocation map), but DBCC finds that the block is not part of any database object. In this example, DBCC reports 32 leaked blocks.

```
                    Stat                                Value                Flags
=====================================|===========================|=====
DBCC Allocation Mode Report
=====================================|===========================|=====
** DBCC Status                       |Errors Detected            |*****
=====================================|===========================|=====
Allocation Summary                   |                           |
=====================================|===========================|=====
   Blocks Total                      |8192                       |
   Blocks in Current Version         |4954                       |
   Blocks in All Versions            |4954                       |
   Blocks in Use                     |4986                       |
```

```
   % Blocks in Use                    |60                         |
** Blocks Leaked                      |32                         |*****
                                      |                           |
=====================================|===========================|=====
Allocation Statistics                 |                           |
=====================================|===========================|=====
   Marked Logical Blocks              |8064                       |
   Marked Physical Blocks             |4954                       |
   Marked Pages                       |504                        |
   Blocks in Freelist                 |126553                     |
   Imaginary Blocks                   |121567                     |
   Highest PBN in Use                 |5432                       |
** 1st Unowned PBN                    |452                        |*****
   Total Free Blocks                  |3206                       |
   Usable Free Blocks                 |3125                       |
   % Free Space Fragmented            |2                          |
   Max Blocks Per Page                |16                         |
   1  Block Page Count                |97                         |
   3  Block Page Count                |153                        |
   4  Block Page Count                |14                         |
   ...
   9  Block Hole Count                |2                          |
   16 Block Hole Count                |194                        |
                                      |                           |
   Database Objects Checked           |1                          |
   B-Array Count                      |1                          |
   Blockmap Identity Count            |1                          |
=====================================|===========================|
=====Connection Statistics                 |                           |
```

sp_iqcheckoptions procedure

Function For the connected user, sp_iqcheckoptions displays a list of the current value and the default value of database and server startup options that have been changed from the default.

Syntax **sp_iqcheckoptions**

Permissions None. The DBA sees all options set on a permanent basis for all groups and users and sees temporary options set for DBA. Users who are not DBAs see their own temporary options. All users see nondefault server startup options.

Usage Requires no parameters. Returns one row for each option that has been changed from the default value. The output is sorted by option name, then by user name.

Description

For the connected user, the sp_iqcheckoptions stored procedure displays a list of the current value and the default value of database and server startup options that have been changed from the default. sp_iqcheckoptions considers all Sybase IQ and SQL Anywhere database options. Sybase IQ modifies some SQL Anywhere option defaults, and these modified values become the new default values. Unless the new Sybase IQ default value is changed again, sp_iqcheckoptions does not list the option.

When sp_iqcheckoptions is run, the DBA sees all options set on a permanent basis for all groups and users and sees temporary options set for DBA. Users who are not DBAs see their own temporary options. All users see nondefault server startup options.

Table 7-5: sp_iqcheckoptions columns

Column name	Description
User_name	The name of the user or group for whom the option has been set. At database creation, all options are set for the public group. Any option that has been set for a group or user other than public is displayed.
Option_name	The name of the option.
Current_value	The current value of the option.
Default_value	The default value of the option.
Option_type	"Temporary" for a TEMPORARY option, else "Permanent".

Examples

In these examples, the temporary option APPEND_LOAD is set to ON and the group mygroup has the option MAX_WARNINGS set to 9. The user joel has a temporary value of 55 set for MAX_WARNINGS.

In the first example, sp_iqcheckoptions is run by the DBA.

```
User_name Option_name          Current_value   Default_value   Option_type
DBA       Ansi_update_constr   CURSORS         Off             Permanent
PUBLIC    Ansi_update_constr   Cursors         Off             Permanent
DBA       Append_Load          ON              OFF             Temporary
DBA       Checkpoint_time      20              60              Temporary
DBA       Connection_authent   Company=MyComp;                 Temporary
                               Application=DBTools;Signa
DBA       Login_procedure      DBA.sp_iq_proce sp_login_envir  Permanent
PUBLIC    Login_procedure      DBA.sp_iq_proce sp_login_envir  Permanent
mygroup   Max_Warnings         9               281474976710655 Permanent
DBA       Thread_count         25              0               Temporary
```

In the second example, sp_iqcheckoptions is run by the user joel.

```
User_name Option_name         Current_value   Default_value   Option_type
joel      Ansi_update_constr  CURSORS         Off             Permanent
PUBLIC    Ansi_update_constr  Cursors         Off             Permanent
joel      Checkpoint_time     20              60              Temporary
joel      Connection_authent  Company=MyComp;                 Temporary
                              Application=DBTools;Signa
joel      Login_procedure     DBA.sp_iq_proce sp_login_envir  Permanent
PUBLIC    Login_procedure     DBA.sp_iq_proce sp_login_envir  Permanent
joel      Max_Warnings        55              281474976710655 Temporary
joel      Thread_count          25               0                Temporary
```

sp_iqclient_lookup procedure

Function

Allows a client application to determine the Sybase IQ user account responsible for a particular data stream, as observed in a network analyzer originating from a specific client IP address/port.

Syntax

sp_iqclient_lookup [*'IPaddress'*], [*Port*], [*UserID*]

Parameters

IPaddress Specifies the IP address of the originating client application.

Port Specifies the port number of the originating client application.

UserID Specifies the Sybase IQ user ID.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

The sp_iqclient_lookup procedure takes the client IP address and port number and returns a single row containing Number (the connection ID), IPaddress, Port, and UserID.

```
1> sp_iqclient_lookup '158.76.235.71',3360
2> go

Number   IPaddress       Port    UserID
------   ---------       ----    ------
15       158.76.235.71   3360    rdeniro
```

Optionally, you can pass a third argument to select only the UserID. If no arguments are passed, sp_iqclient_lookup returns all current logins with their IP addresses and port numbers. For example:

```
sp_iqclient_lookup

Number   IPaddress         Port    UserID
------   ---------         ----    ------
11       162.66.131.36     2082    mbrando
```

```
21       162.66.100.233   1863    apacino
22       162.66.100.206   8080    jcaan
23       162.66.100.119   6901    rduvall
24       162.66.100.125   7001    dkeaton
25       162.66.100.124   6347    jcazale

(6 rows affected)
(return status = 0)
```

If a client application is not using TCP/IP or for internal connections, the address appears as 127.0.0.1.

Note This information is available for logged on users only. No historical login data is kept on the server for this purpose.

Side effects

The sp_iqclient_lookup stored procedure may impact server performance, which varies from one installation to another. Finding the login name entails scanning through all current active connections on the server; therefore, the impact may be greater on servers with large numbers of connections. Furthermore, this information cannot be cached as it is dynamic — sometimes highly dynamic. It is, therefore, a matter for the local system administrator to manage the use of this stored procedure, as well as monitor the effects on the server, just as for any other client application that uses server facilities.

Examples

Shows IP addresses for UserID jcazale:

```
sp_iqclient_lookup null, null, jcazale

Number   IPaddress        Port    UserID
------   ----------       ----    ------
11       162.66.131.36    2082    jcazale
15       164.66.131.36    1078    jcazale
```

Shows IP addresses from client IP 162.66.131.36:

```
sp_iqclient_lookup '162.66.131.36'

Number   IPaddress        Port    UserID
------   ----------       ----    ------
11       162.66.131.36    2082    jcazale
12       162.66.131.36    1078    jcaan
```

Note The result is empty when the user specifies an incorrect argument.

---

sp_iqcolumn procedure

Function

Displays information about columns in a database.

Syntax1

sp_iqcolumn ([*table_name*],[*table_owner*], [*table_loc*])

Syntax2

sp_iqcolumn
[table_name=*'table_name'*],[table_owner=*'tableowner'*],[table_loc=*'table_loc'*]

Usage

Syntax1 If you specify *table_owner* without specifying *table_name*, you must substitute NULL for *table_name*. For example, `sp_iqcolumn NULL,DBA`.

Syntax2 The parameters can be specified in any order. Enclose *'table_name'* and *'table_owner'* in single quotes.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Displays information about columns in a database. Specifying the *table_name* parameter returns the columns only from tables with that name. Specifying the *table_owner* parameter returns only tables owned by that user. Specifying both *table_name* and *table_owner* parameters chooses the columns from a unique table, if that table exists. Specifying *table_loc* returns only tables that are defined in that segment type. Specifying no parameters returns all columns for all tables in a database. sp_iqcolumn does not return column information for system tables.

Table 7-6: sp_iqcolumn columns

Column name	Description
table_name	The name of the table
table_owner	The owner of the table
column_name	The name of the column
domain_name	The data type
width	The precision of numeric data types that have precision and scale or the storage width of numeric data types without scale; the width of character data types
scale	The scale of numeric data types
nulls	'Y' if the column can contain NULLS, 'N' if the column cannot contain NULLS
default	'Identity/Autoincrement' if the column is an identity/ autoincrement column, null if not.
cardinality	The distinct count, if known, by indexes
est_cardinality	The estimated number of distinct values, set to 255 automatically if the column was created with the MINIMIZE_STORAGE option ON, or a user-supplied value from the IQ UNIQUE constraint specified in CREATE TABLE
location	TEMP = IQ temporary store, MAIN = IQ main store, SYSTEM = catalog store
isPartitioned	'Y' if the column belongs to a partitioned table and has one or more partitions whose dbspace is different from the table partition's dbspace, 'N' if the column's table is not partitioned or each partition of the column resides in the same dbspace as the table partition.
remarks	User comments added with the COMMENT statement
check	the check constraint expression

Example

The following variations in syntax both return all of the columns in the table Departments:

```
sp_iqcolumn Departments
call sp_iqcolumn (table_name='Departments')
```

```
table_name   table_owner  column_name     domain_name  width  scale  nulls  default
Departments  GROUPO       DepartmentID    integer       4         0  N      (NULL)
Departments  GROUPO       DepartmentName  char         40         0  N      (NULL)
Departments  GROUPO       DepartmentHead  integer       4         0  Y      (NULL)

cardinality  est_cardinality  location  isPartitioned  remarks  check
5             5               Main       N             (NULL)   (NULL)
0             5               Main       N             (NULL)   (NULL)
5             5               Main       N             (NULL)   (NULL)
```

The following variation in syntax returns all of the columns in all of the tables owned by table owner DBA.

```
sp_iqcolumn table_owner='DBA'
```

sp_iqcolumnuse procedure

Function

Reports detailed usage information for columns accessed by the workload.

Syntax

sp_iqcolumnuse

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Columns from tables created in SYSTEM are not reported.

Table 7-7: sp_iqcolumnuse columns

Column name	Description
TableName	Table name
ColumnName	Column name
Owner	Username of column owner
UID**	Column Unique Identifier
LastDT	Date/time of last access
NRef	Number of query references

**UID is a number assigned by the system that uniquely identifies the instance of the column (where instance is defined when an object is created).

Example

Sample output from the sp_iqcolumnuse procedure:

```
TableName       ColumnName      Owner   UID            LastDT              NRef
orders          o_orderdate     DBA                151  20070917 22:41:22..  13
orders          o_shippriority  DBA                154  20070917 22:41:22..  13
lineitem        l_orderkey      DBA                186  20070917 22:41:22..  13
lineitem        l_extendedp..   DBA                191  20070917 22:41:22..  13
lineitem        l_discount      DBA                192  20070917 22:41:22..  13
lineitem        l_shipdate      DBA                196  20070917 22:41:22..  13
```

```
#tmp1          expression    DBA    10000000001218   20070917 22:57:36..   1
#tmp1          expression    DBA    10000000001222   20070917 22:41:58..   1
...
```

Note The long numbers in the example above are temporary IDs.

See also

“Monitoring workloads,” in Chapter 3, “Optimizing Queries and Deletions,” in the *Performance and Tuning Guide*.

“sp_iqdbspace procedure” on page 388, “sp_iqindexadvice procedure” on page 418, “sp_iqindexuse procedure” on page 426, “sp_iqtableuse procedure” on page 477, “sp_iqunusedcolumn procedure” on page 481, “sp_iqunusedindex procedure” on page 482, “sp_iqunusedtable procedure” on page 483, and “sp_iqworkmon procedure” on page 490

“INDEX_ADVISOR option” in Chapter 2, “Database Options,” in *Reference: Statements and Options*

sp_iqconnection procedure

Function

Shows information about connections and versions, including which users are using temporary dbspace, which users are keeping versions alive, what the connections are doing inside Sybase IQ, connection status, database version status, and so on.

Syntax

sp_iqconnection [*connhandle*]

Usage

The input parameter *connhandle* is equal to the Number connection property and is the ID number of the connection. The connection_property system function returns the connection ID:

```
SELECT connection_property ( 'Number' )
```

When called with an input parameter of a valid *connhandle*, sp_iqconnection returns the one row for that connection only.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

sp_iqconnection returns a row for each active connection. The columns ConnHandle, Name, Userid, LastReqTime, ReqType, CommLink, NodeAddr, and LastIdle are the connection properties Number, Name, Userid, LastReqTime, ReqType, CommLink, NodeAddr, and LastIdle respectively, and return the same values as the system function sa_conn_info. The additional columns return connection data from the Sybase IQ side of the Sybase IQ engine. Rows are ordered by ConnCreateTime.

The column MPXServerName stores information related to multiplex internode communication (INC), as shown in Table 7-8:

Table 7-8: MPXServerName column values

Server where run	MPXServerName column content
Simplex server	NULL (all connections are local/user connections).
Multiplex coordinator	• NULL for local/user connections. • Contains value of secondary node's server name (source of connection) for every INC connection (either on-demand or dedicated heartbeat connection.
Multiplex secondary	• NULL for local/user connections. • Contains value of coordinator's server name (source of connection).

For information on multiplex capability, see *Using Sybase IQ Multiplex*.

In Java applications, specify Sybase IQ-specific connection properties from TDS clients in the RemotePWD field. This example, where myconnection becomes the IQ connection name, shows how to specify IQ specific connection parameters:

```
p.put("RemotePWD",",,CON=myconnection");
```

For more details about using the RemotePWD parameter, see "Specifying a database on a server" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - Programming > SQL Anywhere Data Access APIs > SQL Anywhere JDBC driver > Using the jConnect JDBC driver > Supplying a URL to the driver.*

Table 7-9: sp_iqconnection columns

Column name	Description
ConnHandle	The ID number of the connection.
Name	The name of the server.
Userid	The user ID for the connection.

Column name	Description
LastReqTime	The time at which the last request for the specified connection started.
ReqType	A string for the type of the last request.
IQCmdType	The current command executing on the Sybase IQ side, if any. The command type reflects commands defined at the implementation level of the engine. These commands consists of transaction commands, DDL and DML commands for data in the IQ store, internal IQ cursor commands, and special control commands such as OPEN and CLOSE DB, BACKUP, RESTORE, and others.
LastIQCmdTime	The time the last IQ command started or completed on the IQ side of the Sybase IQ engine on this connection.
IQCursors	The number of cursors open in the IQ store on this connection.
LowestIQCursorState	The IQ cursor state, if any. If multiple cursors exist on the connection, the state displayed is the lowest cursor state of all the cursors; that is, the furthest from completion. Cursor state reflects internal Sybase IQ implementation detail and is subject to change in the future. For this version, cursor states are: NONE, INITIALIZED, PARSED, DESCRIBED, COSTED, PREPARED, EXECUTED, FETCHING, END_OF_DATA, CLOSED and COMPLETED. As suggested by the names, cursor state changes at the end of the operation. A state of PREPARED, for example, indicates that the cursor is executing.
IQthreads	The number of Sybase IQ threads currently assigned to the connection. Some threads may be assigned but idle. This column can help you determine which connections are using the most resources.
TxnID	The transaction ID of the current transaction on the connection. This is the same as the transaction ID displayed in the *.iqmsg* file by the BeginTxn, CmtTxn, and PostCmtTxn messages, as well as the Txn ID Seq logged when the database is opened.
ConnCreateTime	The time the connection was created.
TempTableSpaceKB	The number of kilobytes of IQ temporary store space in use by this connection for data stored in IQ temp tables.
TempWorkSpaceKB	The number of kilobytes of IQ temporary store space in use by this connection for working space such as sorts, hashes, and temporary bitmaps. Space used by bitmaps or other objects that are part of indexes on Sybase IQ temporary tables are reflected in TempTableSpaceKB.
IQConnID	The 10-digit connection ID displayed as part of all messages in the *.iqmsg* file. This is a monotonically increasing integer unique within a server session.
satoiq_count	An internal counter used to display the number of crossings from the SQL Anywhere side to the IQ side of the Sybase IQ engine. This might be occasionally useful in determining connection activity. Result sets are returned in buffers of rows and do not increment satoiq_count or iqtosa_count once per row.
iqtosa_count	An internal counter used to display the number of crossings from the IQ side to the SQL Anywhere side of the Sybase IQ engine. You might find this column to be occasionally useful in determining connection activity.
CommLink	The communication link for the connection. This is one of the network protocols supported by Sybase IQ, or is local for a same-machine connection.

Column name	Description
NodeAddr	The node for the client in a client/server connection.
LastIdle	The number of ticks between requests.
MPXServerName	If an INC connection, the varchar(128) value contains the name of the multiplex server where the INC connection originates. NULL if not an INC connection.

Example

An example of sp_iqconnection output:

```
ConnHandle   Name                Userid   LastReqTime                        ReqType
==========   =================   ======   ========================           =======
         9   'IQ_MPX_SERVER_H'   'dbo'    '2008-11-18 13:15:00.035'          'EXEC'
        11   'IQ_MPX_SERVER_H'   'dbo'    '2008-11-18 13:15:00.046'          'EXEC'
          13   'IQ_MPX_SERVER_H'  'dbo'    '2008-11-18 14:52:55.003'          'EXEC'
          15   'IQ_MPX_SERVER_H'  'dbo'    '2008-11-18 14:53:25.005'          'EXEC'
          17   'SQL_DBC_49450e8   'DBA'    '2008-11-18 14:59:45.680'          'OPEN'
          44   'Sybase Central 1  'DBA'    '2008-11-18 14:59:45.023'          'CLOSE

IQCmdType              LastIQCmdTime              IQCursors   LowestIQCursorState
=========              ======================     =========   ===================
'NONE'                 2008-11-18 13:15:00.0              0   'NONE'
'NONE'                 2008-11-18 13:15:00.0              0   'NONE'
'NONE'                 2008-11-18 14:52:55.0              0   'NONE'
'NONE'                 2008-11-18 14:53:25.0              0   'NONE'
'IQUTILITYOPENCURSOR'  2008-11-18 14:59:45.0              0   'NONE'
'NONE'                 2008-11-18 14:43:33.0              0   'NONE'


IQthreads  TxnID  ConnCreateTime                 TempTableSpaceKB TempWorkSpaceKB
=========  =====  ====================           ================ ==============
1            0    2008-11-18 13:14:09.0                 0           0
1            0    2008-11-18 13:14:34.0                 0           0
1            0    2008-11-18 13:14:55.0                 0           0
1            0    2008-11-18 13:15:25.0                 0           0
1          50024  2008-11-18 13:28:08.0                 0           0
1          50545  2008-11-18 14:03:50.0                 0           0


IQconnID  satoiq_count  iqtosa_count  CommLink  NodeAddr   LastIdle   MPXServerName
========  ============  ============  ========  ========   =========  ===========
23198               28            12  'local'         ''      2977  'mpx0631_r1'
23202               28            12  'local'         ''      1503  'mpx0631_r2'
23207              127            12  'local'         ''     10000  'mpx0631_w1'
23212              127            12  'local'         ''     10000  'mpx0631_w2'
23267              658            66  'TCPIP''10.18.60.181'   9375
23443              510            54  'local'         ''     1238
```

sp_iqconstraint procedure

Function
Lists referential integrity constraints defined using CREATE TABLE or ALTER TABLE for the specified table or column.

Syntax

sp_iqconstraint ['*table-name*', '*column-name*', '*table-owner*']

Permissions
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description
If table name and column name are omitted, reports all referential integrity constraints for all tables including temporary ones in the current connected database. The information includes unique or primary key constraint, referential constraint, and associated role name that are defined by the CREATE TABLE and/or ALTER TABLE statements.

Example
This is sample output that displays all primary key/foreign key pairs where either the candidate key or foreign key contains column ck1 for owner bob in all tables:

```
call sp_iqconstraint('','ck1','bob')
```

```
PTAB1 bob ASIQ_IDX_T27_HG  unique   ck1,ck2  selftab bob CK6FK3  Y
ASIQ_IDX_T42_HG  ck1,ck2

PTAB2 bob ASIQ_IDX_T27_HG  unique   ck1,ck2  selftab bob CK6FK4  Y
ASIQ_IDX_T206_I42_HG  ck1,ck2

selftab bob ASIQ_IDX_T26_HG  unique   ck1,ck2  selftab bob CK3FK1  Y
ASIQ_IDX_T206_I42_HG  ck1,ck2
```

The columns displayed are:

- Primary enforced table
- Table owner
- Candidate key index
- Primary key or Unique
- Primary key columns
- Foreign table
- Foreign table owner
- Foreign key role name
- Enforced status (“Y” for enforced, “N” for unenforced)
- Foreign key index
- Foreign key columns

- Location ("TEMP," "MAIN," or "SYSTEM")

sp_iqcontext procedure

Function
: Tracks and displays, by connection, information about statements that are currently executing.

Syntax
: **sp_iqcontext** [*connhandle*]

Usage
: The input parameter *connhandle* is equal to the Number connection property and is the ID number of the connection.

When called with an input parameter of a valid *connhandle*, sp_iqcontext returns the information only for that connection.

Permissions
: DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description
: sp_iqcontext lets the DBA determine what statements are running on the system at any given moment, and identify the user and connection that issued the statement. With this information, you can use this utility to:

- Match the statement text with the equivalent line in sp_iqconnection to get resource usage and transactional information about each connection
- Match the statement text to the equivalent line in the SQL log created when the -zr server option is set to ALL or SQL
- Use connection information to match the statement text in sp_iqcontext to the equivalent line in the *.iqmsg* file, which includes the query plan, when Sybase IQ can collect it
- Match statement text to an IQ stack trace (*stktrc-yyyymmdd-hhnnss_#.iq*), if one is produced
- Collate this information with an operating system stack trace that might be produced, such as pstack on Sun Solaris

The maximum size of statement text collected is the page size of the catalog store.

Table 7-10: sp_iqcontext columns

Column name	Description
ConnOrCursor	CONNECTION or CURSOR.
ConnHandle	The ID number of the connection.
Name	The name of the server.

Column name	Description
Userid	The user ID for the connection or cursor.
numIQCursors	If column 1 is CONNECTION, the number of cursors open on this connection. If column 1 is CURSOR, a number assigned sequentially to cursors associated with this connection.
IQthreads	The number of IQ threads currently assigned to the connection. Some threads may be assigned but idle.
TxnID	The transaction ID of the current transaction.
ConnOrCurCreateTime	The time this connection or cursor was created.
IQConnID	The 10-digit connection ID displayed as part of all messages in the *.iqmsg* file. This is a monotonically increasing integer unique within a server session.
IQGovernPriority	A value that indicates the order in which the queries of a user are queued for execution. 1 indicates high priority, 2 (the default) medium priority, and 3 low priority. A value of -1 indicates that IQGovernPriority does not apply to the operation. Set the IQGovernPriority value with the database option IQGOVERN_PRIORITY. See “Setting query priority” in Chapter 3, “Optimizing Queries and Deletions” of the *Performance and Tuning Guide*.
CmdLine	First 4096 characters of the user command being executed.

Example

The following example shows an excerpt from output when `sp_iqcontext` is issued with no parameter, producing results for all current connections.

```
CONNECTION 701773517 dba7 DBA 6 1 1324 2009-06-04 09:24:17.000 4 NO COMMAND
CURSOR 701773517 dba7 DBA 1 0 1324 2009-06-04 09:24:46.000 4 2 select * from foo1
CURSOR 701773517 dba7 DBA 2 0 1324 2009-06-04 09:24:47.000 4 2 select a from foo1
...
CURSOR 701773517 dba7 DBA 6 0 1324 2009-06-04 09:24:47.000 4 2 select e from foo1
CONNECTION 1271624950 dba7 DBA 0 12 1377 2009-06-04 09:24:12.000 3 2 sp_iqcheckdb
CONNECTION 1841476383 dba7 DBA 10 1 1337 2009-06-04 09:24:19.000 5 2 call sp_iqcontext()
CURSOR 1841476383 dba7 DBA 1 0 1337 2009-06-04 09:24:47.000 5 2 select * from foo
...
CURSOR 1841476383 dba7 DBA 10 0 1337 2009-06-04 09:24:48.000 5 2 select i from foo
```

The first line of output shows connection 701773517 (IQ connection ID 4). This connection is on server dba7, user DBA. It has six active cursors and one IQ thread, and was created from transaction 1324. This connection was not executing a command when sp_iqcontext was issued. The next six lines of output list cursors in use by this connection (only three are shown here.)

Two connections are running stored procedures. Connection 1271624950 is running sp_iqcheckdb directly from dbisql, has no active cursors but is using 12 IQ threads. Connection 1841476383 has called sp_iqcontext as a procedure, is using only 1 IQ thread, and has 10 active cursors (only the first and last are shown here.) In both cases, the name of the stored procedure appears but not the line of code executing within it.

The connection handle (701773517 for the first connection in this example) identifies results in the -zr log. The IQ connection ID (4 for the first connection in this example) identifies results in the *.iqmsg* file. On UNIX systems, you can use the grep command to locate all instances of the connection handle or connection ID, making it easy to correlate information from all sources. The 2 before the user command fragment indicates that this is a medium priority query.

sp_iqcopyloginpolicy procedure

Function

Creates a new login policy by copying an existing one.

Syntax1

call sp_iqcopyloginpolicy (‘*existing-policy-name*’, ‘*new-policy-name*’)

Syntax2

sp_iqcopyloginpolicy ‘*existing-policy-name*’, ‘*new-policy-name*’

Syntax3

sp_iqcopyloginpolicy *existing-policy-name*, *new-policy-name*
policy_name]

Usage

existing policy name The login policy to copy.

new policy name Name of the new login policy to create (CHAR(128)).

Permissions

Requires DBA authority.

See also

"sp_iqpassword procedure" on page 441

CREATE USER Statement in Chapter 1, "SQL Statements," in *Reference: Statements and Options*

Examples

The following stored procedure creates a new login policy named *lockeduser* by copying the login policy option values from the existing login policy named *root*.

```
call sp_iqcopyloginpolicy ('root','lockeduser')
```

sp_iqcursorinfo procedure

Function

Displays detailed information about cursors currently open on the server.

Syntax

sp_iqcursorinfo [*cursor-name*] [, *conn-handle*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

cursor-name The name of the cursor. If only this parameter is specified, sp_iqcursorinfo returns information about all cursors that have the specified name in all connections.

conn-handle An integer representing the connection ID. If only this parameter is specified, sp_iqcursorinfo returns information about all cursors in the specified connection.

The sp_iqcursorinfo procedure can be invoked without any parameters. If no parameters are specified, sp_iqcursorinfo returns information about all cursors currently open on the server. If both parameters are specified, sp_iqcursorinfo reports information about all of the cursors that have the specified name and are in the specified connection.

If you do not specify the first parameter, but specify the second parameter, you must substitute NULL for the omitted parameter. For example, `sp_iqcursorinfo NULL, 1`.

Table 7-11: sp_iqcursorinfo usage examples

Syntax	Output
sp_iqcursorinfo	Displays information about all cursors currently open on the server
sp_iqcursorinfo ‘cursor1’	Displays information about the all cursors named cursor1 in all connections
sp_iqcursorinfo NULL, 3	Displays information about all cursors in connection 3
sp_iqcursorinfo ‘cursor2’, 4	Displays information about all the cursors named cursor2 in connection 4

Description

The sp_iqcursorinfo stored procedure displays detailed information about cursors currently open on the server. The sp_iqcursorinfo procedure enables database administrators to monitor cursor status using just one stored procedure and view statistics such as how many rows have been updated, deleted, and inserted.

If you specify one or more parameters, the result is filtered by the specified parameters. For example, if *cursor-name* is specified, only information about the specified cursor is displayed. If *conn-handle* is specified, sp_iqcursorinfo returns information only about cursors in the specified connection. If no parameters are specified, sp_iqcursorinfo displays information about all cursors currently open on the server.

The sp_iqcursorinfo procedure returns information in the following columns:

Table 7-12: sp_iqcursorinfo columns

Column name	Description
Name	The name of the cursor
ConnHandle	The ID number of the connection
IsUpd	Y: the cursor is updatable; N otherwise
IsHold	Y: the cursor is a hold cursor; N otherwise
IQConnID	The ten digit connection ID displayed as part of all messages in the *.iqmsg* file. This number is a monotonically increasing integer unique within a server session.
UserID	User ID (or user name) for the user who created and ran the cursor
CreateTime	The time of cursor creation
CurrentRow	The current position of the cursor in the result set
NumFetch	The number of times the cursor fetches a row. The same row can be fetched more than once.
NumUpdate	The number of times the cursor updates a row, if the cursor is updatable. The same row can be updated more than once.

Column name	Description
NumDelete	The number of times the cursor deletes a row, if the cursor is updatable.
NumInsert	The number of times the cursor inserts a row, if the cursor is updatable.
RWTabOwner	The owner of the table that is opened in RW mode by the cursor.
RWTabName	The name of the table that is opened in RW mode by the cursor.
CmdLine	The first 4096 characters of the command the user executed

Example — Display information about all cursors currently open on the server:

```
sp_iqcursorinfo

Name      ConnHandle     IsUpd      IsHold      IQConnID      UserID
----------------------------------------------------------------------
crsr1             1         Y           N           118         DBA
crsr2             3         N           N           118         DBA

CreateTime                 CurrentRow      NumFetch      NumUpdate
------------------------------------------------------------------
2009-06-26 15:24:36.000         19      100000000     200000000
2009-06-26 15:38:38.000      20000      200000000

NumDelete     NumInsert      RWTabOwner      RWTabName         CmdLine
-----------------------------------------------------------------------
 20000000    3000000000             DBA          test1    call proc1()
                                                          call proc2()
```

See also — *Reference: Statements and Options*: DECLARE CURSOR statement [ESQL] [SP], DECLARE CURSOR statement [T-SQL], UPDATE (positioned) statement [ESQL] [SP] and DELETE (positioned) statement [ESQL] [SP], FORCE_NO_SCROLL_CURSORS option, and FORCE_UPDATABLE_CURSORS option

"Using cursors in procedures" in Chapter 1, "Using Procedures and Batches," in the *System Administration Guide: Volume 2*

"Cursors in transactions" in Chapter 10, "Transactions and Versioning" in the *System Administration Guide: Volume 1*

sp_iqdatatype procedure

Function — Displays information about system data types and user-defined data types.

Syntax — **sp_iqdatatype** [*type-name*], [*type-owner*], [*type-type*]

Permissions — DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage — **type-name** The name of the data type.

type-owner The name of the creator of the data type.

type-type The type of data type. Allowed values are:

- SYSTEM: displays information about system defined data types (data types owned by user SYS or dbo) only

- ALL: displays information about user and system data types
- Any other value: displays information about user data types

The sp_iqdatatype procedure can be invoked without any parameters. If no parameters are specified, only information about user-defined data types (data types not owned by dbo or SYS) is displayed by default.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqdatatype NULL, NULL, SYSTEM` and `sp_iqdatatype NULL, user1`.

Table 7-13: sp_iqdatatype usage examples

Syntax	Output
sp_iqdatatype	Displays information about all user-defined data types in the database
sp_iqdatatype address	Displays information about the user-defined data type named address
sp_iqdatatype non_existing_type	No rows returned, as the data type non_existing_type does not exist
sp_iqdatatype NULL, DBA	Displays information about all user-defined data types owned by DBA
sp_iqdatatype address, DBA	Displays information about the data type address owned by DBA
sp_iqdatatype rowid	rowid is a system-defined data type. If there is no user-defined data type also named rowid, no rows are returned. (By default, only user-defined data types are returned.)
sp_iqdatatype rowid, SYS	No rows returned, as the data type rowid is not a user-defined data type (by default, only user-defined data types are returned)
sp_iqdatatype NULL, NULL, SYSTEM	Displays information about all system defined data types (owned by dbo or SYS)
sp_iqdatatype rowid, NULL, SYSTEM	Displays information about the system data type rowid
sp_iqdatatype NULL, NULL, 'ALL'	Displays information about the rowid data type

Description

The sp_iqdatatype stored procedure displays information about system and user-defined data types in a database. User-defined data types are also referred to as domains. Predefined domain names are not included in the sp_iqdatatype output.

If you specify one or more parameters, the sp_iqdatatype result is filtered by the specified parameters. For example, if *type-name* is specified, only information about the specified data type is displayed. If *type-owner* is specified, sp_iqdatatype only returns information about data types owned by the specified owner. If no parameters are specified, sp_iqdatatype displays information about all the user-defined data types in the database.

The sp_iqdatatype procedure returns information in the following columns:

Table 7-14: sp_iqdatatype columns

Column name	Description
type_name	The name of the data type
creator	The owner of the data type
nulls	Y indicates the user-defined data type allows nulls; N indicates the data type does not allow nulls.
width	Displays the length of string columns, the precision of numeric columns, and the number of bytes of storage for all other data types
scale	Displays the number of digits after the decimal point for numeric data type columns and zero for all other data types
"default"	The default value for the data type
"check"	The CHECK condition for the data type

Example

Display information about the user-defined data type address:

```
sp_iqdatatype address

type_name    creator    nulls    width    scale    "default"    "check"
address      DBA        Y        5        0         (NULL)       (NULL)
```

See also

CREATE DOMAIN statement in *Reference: Statements and Options*

Chapter 3, "SQL Data Types"

sp_iqdbsize procedure

Function

Displays the size of the current database.

Syntax

sp_iqdbsize(
[**main**]
)

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Returns the total size of the database. Also returns the number of pages required to hold the database in memory and the number of IQ pages when the database is compressed (on disk).

Table 7-15: sp_iqdbsize columns

Column name	Description
Database	The path name of the database file.
Physical Blocks	Total database size in blocks.
KBytes	Total database size in kilobytes.
Pages	Total number of IQ pages.
Compressed Pages	Total number of IQ pages that are compressed (on disk). Subset of Pages.
NBlocks	Total size, in IQ blocks, used to store the data in tables and join indexes.
Catalog Blocks	Total size, in IQ blocks, used to store the metadata for tables and join indexes. Subset of NBlocks.

Descriptions of sp_iqdbsize columns:

Database The path name of the current database file.

Physical Blocks An IQ database consists of one or more dbspaces. Each dbspace has a fixed size, which is originally specified in units of megabytes. This megabyte quantity is converted to blocks using the IQ page size and the corresponding block size for that IQ page size. The Physical Blocks column reflects the cumulative total of each Sybase IQ dbspace size, represented in blocks.

For the correspondence between IQ page size and block size, see Chapter 4, “Managing System Resources” in the *Performance and Tuning Guide*.

KBytes The total size of the database in kilobytes. This value is the total size of the database in blocks (Physical Blocks in the previous sp_iqdbsize column) multiplied by the block size. The block size depends on the IQ page size.

Pages The total number of IQ pages necessary to represent in memory all of the data stored in tables and join indexes, as well as the metadata for these objects. This value is always greater than or equal to the value of Compressed Pages (the next sp_iqdbsize column).

Compressed Pages The total number of IQ pages necessary to store on disk the data in tables and join indexes as well as the metadata for these objects. This value is always less than or equal to the value of Pages (the previous sp_iqdbsize column), because Sybase IQ compresses pages when the IQ page is written from memory to disk. The sp_iqdbsize Compressed Pages column represents the number of compressed pages.

NBlocks The total size in blocks used to store the data in tables and join indexes. This value is always less than or equal to the sp_iqdbsize Physical Blocks value.

Catalog Blocks The total size in blocks used to store the metadata for tables and join indexes.

Example

Displays size information for the database iqdemo:

```
sp_iqdbsize
```

```
Database
PhysicalBlocks KBytes Pages CompressedPages NBlocks CatalogBlocks
============== ====== ===== =============== ======= =============
/system1/sybase/IQ-15_1/demo/iqdemo.db
          1280    522   688            257    1119            18
```

See also

“Specifying page size” in “Overview of memory use” in Chapter 4, “Managing System Resources” in the *Performance and Tuning Guide*

“Working with database objects” in Chapter 5, “Working with Database Objects” in the *System Administration Guide: Volume 1*

sp_iqdbspace procedure

Function

Displays detailed information about each IQ dbspace.

Syntax

sp_iqdbspace [*dbspace-name*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

You can use the information from iqdbspace to determine whether data must be moved, and for data that has been moved, whether the old versions have been deallocated. sp_iqdbspace displays this information:

Table 7-16: sp_iqdbspace columns

Column name	Description
DBSpaceName	Name of the dbspace as specified in the CREATE DBSPACE statement. Dbspace names are case-insensitive for databases created with CASE RESPECT.
DBSpaceType	Type of the dbspace (MAIN or TEMPORARY only).
Writable	T (writable) or F (not writable).
Online	T (online) or F (offline).
Usage	Percent of dbspace currently in use by all files in the dbspace.
TotalSize	Total size of all files in the dbspace in the units B (bytes), K (kilobytes), M (megabytes), G (gigabytes), T (terabytes), or P (petabytes).
Reserve	Total reserved space that can be added to all files in the dbspace.
NumFiles	Number of files in the dbspace.
NumRWFiles	Number of read/write files in the dbspace.
Stripingon	T (On) or F (Off).
StripeSize	Amount of data written to the dbspace before moving to the next dbspace, if disk striping is on.
BlkTypes	Space used by both user data and internal system structures (see Table 7-17).
OkToDrop	‘Y’ indicates the dbspace can be dropped; otherwise ‘N’.

Table 7-17 lists the values of the block type identifiers.

Table 7-17: sp_iqdbspace block types

Identifier	Block type
A	Active Version
B	Backup Structures
C	Checkpoint Log
D	Database Identity
F	Free list
G	Global Free list Manager
H	Header Blocks of the free list
I	Index advice storage
M	Multiplex CM*
O	Old Version
T	Table use
U	Index use
N	Column use
X	Drop at checkpoint

*The multiplex commit identity block (actually 128 blocks) exists in all IQ databases, even though it is not used by simplex databases. For information on multiplex capability, see *Using Sybase IQ Multiplex*.

Example

The following output displays information about dbspaces.

```
sp_iqdbspace;
```

DBSpace Name	DBSpaceType	Writable	Online	Usage	TotalSize	Reserve	NumFiles	NumRWFiles	Stripingon	StripeSize	BlkTypes	OkToDrop
IQ_MAIN	MAIN	T	T	55	75M	200M	1	1	T	1K	1H, 5169A, 190	N
IQ_SYSTEM_MAIN	MAIN	T	T	21	300M	50M	1	1	F	8K	1H, 7648F, 32D, 128M	N
IQ_SYSTEM_TEMP	TEMPORARY	T	T	1	100M	50M	1	1	F	8K	1H, 64F, 32A	N

See also

"sp_iqdbspaceinfo procedure" on page 391

"sp_iqdbspaceobjectinfo procedure" on page 394

"sp_iqindexinfo procedure" on page 421

sp_iqdbspaceinfo procedure

Function

Displays the size of each object and subobject used in the specified table or join index.

Syntax

sp_iqdbspaceinfo [*dbspace-name*] [, *owner_name*] [, *object_name*] [, *object-type*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

dbspace_name If specified, sp_iqdbspaceinfo displays one line for each table that has any component in the specified dbspace. Otherwise, the procedure shows information for all dbspaces in the database.

owner_name Owner of the object. If specified, sp_iqdbspaceinfo displays output only for tables and join indexes with the specified owner. If not specified, sp_iqdbspaceinfo displays information on tables and join indexes for all users in the database.

object_name Name of the table or join index . If not specified, sp_iqdbspaceinfo displays information on all tables and join indexes in the database.

object_type Valid object types are table (for table information) or joinindex (for join index information). If unspecified, object type defaults to table.

All parameters are optional and any parameter may be supplied independent of another parameter's value.

The sp_iqdbspaceinfo stored procedure supports wildcard characters for interpreting *dbspace_name*, *object_name*, and *owner_name*. It shows information for all dbspaces that match the given pattern in the same way the LIKE clause matches patterns inside queries.

Description

sp_iqdbspaceinfo shows the DBA the amount of space used by objects that reside on each dbspace. The DBA can use this information to determine which objects must be relocated before a dbspace can be dropped. The sub-object columns display sizes reported in integer quantities followed by the suffix B, K, M, G, T, or P, representing bytes, kilobytes, megabytes, gigabytes, terabytes, and petabytes, respectively.

For tables, sp_iqdbspaceinfo displays sub-object sizing information for all sub-objects (using integer quantities with the suffix B, K, M, G, T, or P). For join indexes, the procedure displays sizing information for the join index and all of its associated sub-objects. The output is sorted by *dbspace_name*, *object_name*, and *owner_name*.

Table 7-18: sp_iqdbspaceinfo columns

Column name	Description
dbspace_name	Name of the dbspace.
object_type	Type of the object (table or joinindex only).
owner	Name of the owner of the object.
object_name	Name of the object (of type tables and join indexes only) located on the dbspace.
object_id	Global object ID of the object.
id	Table id or join-index ID of the object.
columns	Size of column storage space on the given dbspace.
indexes	Size of index storage space on the given dbspace. Does not use sytem-generated indexes (for example, HG indexes in unique constraints or FP indexes).
metadata	Size of storage space for metadata objects on the given dbspace.
primary_key	Size of storage space for primary key related objects on the given dbspace.
unique_constraint	Size of storage space for unique constraint-related objects on the given dbspace.
foreign_key	Size of storage space for foreign-key-related objects on the given dbspace.

Note
If you run sp_iqdbspaceinfo against a server you have started with the -r switch (read-only), you see the error `Msg 13768, Level 14, State 0: SQL Anywhere Error -757: Modifications not permitted for read-only database`. This behavior is expected. The error does not occur on other stored procedures such as sp_iqdbspace, sp_iqfile, sp_iqdbspaceobjectinfo or sp_iqobjectinfo.

Examples

Displays the size of all objects and sub-objects in all tables in all dbspaces in the database:

```
sp_iqdbspaceinfo
```

```
dbspace_name    object_type  owner   object_name    object_id  id   columns
```

iq_main	table	DBA	emp1	3689	741	96K
iq_main	table	DBA	iq_dummy	3686	740	24K
iq_main	table	DBA	sale	3698	742	96K
iq_main	table	GROUPO	Contacts	3538	732	288K
iq_main	table	GROUPO	Customers	3515	731	240K
iq_main	table	GROUPO	Departments	3632	738	72K
iq_main	table	GROUPO	Employees	3641	739	408K
iq_main	table	GROUPO	FinancialCodes	3612	736	72K
iq_main	table	GROUPO	FinancialData	3621	737	96K
iq_main	table	GROUPO	Products	3593	735	272K
iq_main	table	GROUPO	SalesOrderItems	3580	734	120K
iq_main	table	GROUPO	SalesOrders	3565	733	144K

indexes	metadata	primary_key	unique_constraint	foreign_key
0B	1.37M	0B	0B	0B
0B	464K	0B	0B	0B
0B	1.22M	0B	0B	0B
0B	5.45M	24K	0B	48K
48K	4.63M	24K	0B	0B
0B	1.78M	24K	0B	48K
0B	8.03M	24K	0B	48K
0B	1.53M	24K	0B	0B
0B	2.19M	24K	0B	48K
192K	4.67M	24K	0B	0B
0B	2.7M	24K	0B	104K
0B	3.35M	24K	0B	144K

Displays the size of all objects and sub-objects owned by a specified user in a specified dbspace in the database:

```
sp_iqdbspaceinfo iq_main,GROUPO
```

dbspace_name	object_type	owner	object_name	object_id	id	columns
iq_main	table	GROUPO	Contacts	3538	732	288K
iq_main	table	GROUPO	Customers	3515	731	240K
iq_main	table	GROUPO	Departments	3632	738	72K
iq_main	table	GROUPO	Employees	3641	739	408K
iq_main	table	GROUPO	FinancialCodes	3612	736	72K
iq_main	table	GROUPO	FinancialData	3621	737	96K
iq_main	table	GROUPO	Products	3593	735	272K
iq_main	table	GROUPO	SalesOrderItems	3580	734	120K
iq_main	table	GROUPO	SalesOrders	3565	733	144K

indexes	metadata	primary_key	unique_constraint	foreign_key
0B	5.45M	24K	0B	48K
48K	4.63M	24K	0B	0B
0B	1.78M	24K	0B	48K
0B	8.03M	24K	0B	48K
0B	1.53M	24K	0B	0B
0B	2.19M	24K	0B	48K
192K	4.67M	24K	0B	0B

```
0B       2.7M      24K         0B                  104K
0B       3.35M     24K         0B                  144K
```

Displays the size of a specified object and its sub-objects owned by a specified user in a specified dbspace in the database:

```
sp_iqdbspaceinfo iq_main,GROUPO,Departments
```

```
dbspace_name    object_type  owner   object_name     object_id  id   columns
iq_main         table        GROUPO  Departments       3632     738   72K

indexes  metadata  primary_key  unique_constraint  foreign_key
0B       1.78M     24K          0B                 48K
```

See also

"sp_iqdbspace procedure" on page 388 and "sp_iqindexinfo procedure" on page 421

sp_iqdbspaceobjectinfo procedure

Function

Lists objects of type table and join index and their sub-objects (columns, indexes, metadata, primary keys, unique constraints, foreign keys, and partitions) for a given dbspace.

Syntax

sp_iqdbspaceobjectinfo [*dbspace-name*] [, *owner_name*] [, *object_name*] [, *object-type*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

dbspace-name If specified, sp_iqdbspaceobjectinfo displays output only for the specified dbspace. Otherwise, it shows information for all dbspaces in the database.

owner-name Owner of the object. If specified, sp_iqdbspaceobjectinfo displays output only for tables and join indexes with the specified owner. If not specified, sp_iqdbspaceobjectinfo displays information for tables and join indexes for all users in the database.

object-name Name of the table or join index. If not specified, sp_iqdbspaceobjectinfo displays information for all tables and join indexes in the database.

object-type Valid object types are table (for table information) or joinindex (for join index information). If unspecified, object type defaults to table.

All parameters are optional and any parameter may be supplied independent of the value of other parameters.

The sp_iqdbspaceobjectinfo stored procedure supports wildcard characters for interpreting *dbspace_name*, *object_name*, and *owner_name*. It displays information for all dbspaces that match the given pattern in the same way as the LIKE clause matches patterns inside queries.

Description

For tables, sp_iqdbspaceobjectinfo displays summary information for all associated sub-objects. For join indexes, it displays sizing information for the join index and for all of its associated sub-objects. The output of the stored procedure is sorted by dbspace_name, owner and object_name.

sp_iqdbspaceobjectinfo displays the following information, based on the input parameter values:

Table 7-19: sp_iqdbspaceobjectinfo columns

Column name	Description
dbspace_name	Name of the dbspace.
object_type	Table or join index.
owner	Name of the owner of the object.
object_name	Name of the object (of type tables and join indexes only) located on the dbspace.
object_id	Global object id of the object.
id	Table id or join-index id of the object.
columns	Number of table columns which are located on the given dbspace. If a column or one of the column-partitions is located on a dbspace, it is counted to be present on that dbspace. The result is displayed in the form n/N (n out of total N columns of the table are on the given dbspace).
indexes	Number of user defined indexes on the table which are located on the given dbspace. Displayed in the form n/N (n out of total N indexes on the table are on the given dbspace). This does not contain indexes which are system generated, such as FP indexes and HG indexes in the case of unique constraints.
metadata	Boolean field (Y/N) to denote if the metadata information of the sub-object is also located on this dbspace.
primary_key	Boolean field (1/0) to denote if the primary key of the table, if any, is located on this dbspace.
unique_constraint	Number of unique constraints on the table which are located on the given dbspace. Displayed in the form n/N (n out of total N unique constraints on the table are in the given dbspace).
foreign_key	Number of foreign_keys on the table which are located on the given dbspace. Displayed in the form n/N (n out of total N foreign keys on the table are in the given dbspace).

Column name	**Description**
partitions	Number of partitions of the table which are located on the given dbspace. Displayed in the form n/N (n out of total N partitions of the table are in the given dbspace).

Examples

Displays information about a specific dbspace in the database:

```
sp_iqdbspaceobjectinfo iq_main
```

```
dbspace_name    object_type  owner   object_name     object_id   id    columns
iq_main         table        DBA     emp1                 3689   741   4/4
iq_main         table        DBA     iq_dummy             3686   740   1/1
iq_main         table        DBA     sale                 3698   742   4/4
iq_main         table        GROUPO  Contacts             3538   732   12/12
iq_main         table        GROUPO  Customers            3515   731   10/10
iq_main         table        GROUPO  Departments          3632   738   3/3
iq_main         table        GROUPO  Employees            3641   739   21/21
iq_main         table        GROUPO  FinancialCodes       3612   736   3/3
iq_main         table        GROUPO  FinancialData        3621   737   4/4
iq_main         table        GROUPO  Products             3593   735   8/8
iq_main         table        GROUPO  SalesOrderItems      3580   734   5/5
iq_main         table        GROUPO  SalesOrders          3565   733   6/6

indexes  metadata  primary_key  unique_constraint  foreign_key  partitions
0/0      Y                   0  0/0                0/0          0/0
0/0      Y                   0  0/0                0/0          0/0
0/0      Y                   0  0/0                0/0          0/0
0/0      Y                   1  0/0                1/1          0/0
1/1      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                1/1          0/0
0/0      Y                   1  0/0                1/1          0/0
0/0      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                1/1          0/0
4/4      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                2/2          0/0
0/0      Y                   1  0/0                3/3          0/0
```

Displays information about the objects owned by a specific user in a specific dbspace in the database:

```
sp_iqdbspaceobjectinfo iq_main,GROUPO
```

```
dbspace_name    object_type  owner   object_name     object_id   id    columns
iq_main         table        GROUPO  Contacts             3538   732   12/12
iq_main         table        GROUPO  Customers            3515   731   10/10
iq_main         table        GROUPO  Departments          3632   738   3/3
iq_main         table        GROUPO  Employees            3641   739   21/21
iq_main         table        GROUPO  FinancialCodes       3612   736   3/3
iq_main         table        GROUPO  FinancialData        3621   737   4/4
iq_main         table        GROUPO  Products             3593   735   8/8
```

```
iq_main         table        GROUPO  SalesOrderItems   3580   734   5/5
iq_main         table        GROUPO  SalesOrders       3565   733   6/6

indexes  metadata  primary_key  unique_constraint  foreign_key  partitions
0/0      Y                   1  0/0                1/1          0/0
1/1      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                1/1          0/0
0/0      Y                   1  0/0                1/1          0/0
0/0      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                1/1          0/0
4/4      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                2/2          0/0
0/0      Y                   1  0/0                3/3          0/0
```

Uses sp_iqdbspaceobjectinfo to construct commands that can be used to move objects. In this example, the commands move all tables on dbspace_x to dbspace_y.

```
SELECT 'ALTER TABLE ' || owner || '.' ||
object_name || ' MOVE TO dbspace_y;'
FROM sp_iqdbspaceobjectinfo()
WHERE object_type = 'table' AND
dbspace_name = 'dbspace_x';
```

The following ALTER TABLE commands are the result:

```
ALTER TABLE DBA.dt1 MOVE TO dbspace_y;
ALTER TABLE DBA.dt2 MOVE TO dbspace_y;
ALTER TABLE DBA.dt3 MOVE TO dbspace_y;
```

See also

"sp_iqdbspace procedure" on page 388 and "sp_iqindexinfo procedure" on page 421

sp_iqdbstatistics procedure

Function

Reports results of the most recent sp_iqcheckdb.

Syntax

sp_iqdbstatistics

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Displays the database statistics collected by the most recent execution of sp_iqcheckdb.

Example

The following example shows the output from sp_iqdbstatistics. For this example, the most recent execution of sp_iqcheckdb was the command sp_iqcheckdb 'allocation database'.

```
            DB Statistics                          Value              Flags
=====================================|===========================|=====
DBCC Allocation Mode Report          |                           |
=====================================|===========================|=====
** DBCC Status                       |Errors Detected            |*****
   DBCC Work units Dispatched        |163                        |
   DBCC Work units Completed         |163                        |
=====================================|===========================|=====
Allocation Summary                   |                           |
=====================================|===========================|=====
   Blocks Total                      |8192                       |
   Blocks in Current Version         |4954                       |
   Blocks in All Versions            |4954                       |
   Blocks in Use                     |4986                       |
   % Blocks in Use                   |60                         |
** Blocks Leaked                     |32                         |*****
                                     |                           |
=====================================|===========================|=====
Allocation Statistics                |                           |
=====================================|===========================|=====
   Blocks Created in Current TXN     |382                        |
   Blocks To Drop in Current TXN     |382                        |
   Marked Logical Blocks             |8064                       |
   Marked Physical Blocks            |4954                       |
   Marked Pages                      |504                        |
   Blocks in Freelist                |126553                     |
   Imaginary Blocks                  |121567                     |
   Highest PBN in Use                |5432                       |
** 1st Unowned PBN                   |452                        |*****
   Total Free Blocks                 |3206                       |
   Usable Free Blocks                |3125                       |
   % Free Space Fragmented           |2                          |
   Max Blocks Per Page               |16                         |
   1  Block Page Count               |97                         |
   3  Block Page Count               |153                        |
   4  Block Page Count               |14                         |
   ...
   9  Block Hole Count               |2                          |
   16 Block Hole Count               |194                        |
                                     |                           |
   Database Objects Checked          |1                          |
   B-Array Count                     |1                          |
   Blockmap Identity Count           |1                          |
=====================================|===========================|=====
Connection Statistics                |                           |
=====================================|===========================|=====
```

See also

For more information on the use of sp_iqcheckdb and the interpretation of the sp_iqcheckdb output, see Chapter 13, “System Recovery and Database Repair,” in the *System Administration Guide: Volume 1*.

sp_iqdroplogin procedure

Function: Drops a Sybase IQ user account.

Syntax1: **call sp_iqdroplogin** (‘*userid*’)

Syntax2: **sp_iqdroplogin** ‘*userid*’

Syntax3: **sp_iqdroplogin** *userid*

Syntax4: **sp_iqdroplogin** (‘*userid*’)

Permissions: DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage: **userid** User ID of the user to drop.

Description: sp_iqdroplogin drops the specified user.

Examples: The following stored procedure calls remove the user rose.

```
sp_iqdroplogin 'rose'

sp_iqdroplogin rose

call sp_iqdroplogin ('rose')
```

See also: “sp_iqaddlogin procedure” on page 355

REVOKE statement in *Reference: Statements and Options*

Chapter 8, “Managing User IDs and Permissions,” in *System Administration Guide: Volume 1*

sp_iqemptyfile procedure

Function: Empties a dbspace file and moves the objects in the file to another available read-write dbpace file.

Syntax: **sp_iqemptyfile** (*logical-file--name*)

Permissions: DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

sp_iqemptyfile empties a dbspace file. The dbspace must be read-only before you can execute the sp_iqemptyfile procedure. The procedure moves the objects in the file to another available read-write dbspace file. If there is not other read-write dbspace file available, then Sybase IQ displays an error message.

Note In a multiplex environment, you can run sp_iqemptyfile only on the coordinator.

There must be one read-write dbspace available in order for the procedure to succeed.

Example

The following empties dbspace dbspace1:

```
sp_iqemptyfile 'dbspace1'
```

sp_iqestjoin procedure

Function

Estimates the space needed to create join indexes for the tables you specify.

Syntax

sp_iqestjoin (*table1_name, table1_row_#, table2_name, table2_row_#, relation, iq_page_size*)

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Returns the amount of space a join index uses based on the tables being joined. This procedure assumes that the database was created with the default block size for the specified IQ page size (or else the estimate is incorrect).

If you specify unqualified table names, then ensure that you are the owner of the tables being joined. If you are not the table owner, then provide a qualified table name for each table, such as 'owner.tablename'.

Table 7-20 lists the sp_iqestjoin parameters.

Table 7-20: sp_iqestjoin parameters

Name	Datatype	Description
table1_name	char(256)	Name of the first table in the join.
table1_row_#	int	Number of rows in the first table that participates in the join.
table2_name	char(256)	Name of the second table in the join.
table2_row_#	int	Number of rows in the second table that participates in the join.
relation	char(9)	Type of join, which can be “one>>many” or “one>>one” (do not leave any spaces between the words and the operator). The default is “one>>many”.
iq_page_size	smallint	The page size defined for the IQ segment of the database (must be a power of 2 between 1024 and 524288; the default is 131072).

Example

```
call sp_iqestjoin ( 'Customers', 1500000,
'SalesOrders', 15000000, 'one>>many', 65536 )
```

Cases	Indexsize	Create time	Msg
Table1:Customers			
Rows: 1500000			
Columns:			
8			
Width:			
223			
Table2: SalesOrders			
Rows: 15000000			
Columns:			
9			
Width:			
134			
IQpagesize:			
65536			
Min Case	48001024	3h0m/CPU	
Max Case	95449088	9h6m/CPU	
Avg Case	70496256	5h53m/CPU	

sp_iqestdbspaces procedure

Function

Estimates the number and size of dbspaces needed for a given total index size.

Syntax

sp_iqestdbspaces (*db_size_in_bytes, iq_page_size, min_#_of_bytes, max_#_of_bytes*)

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Displays information about the number and size of dbspace segments based on the size of the database, the IQ page size, and the range of bytes per dbspace segment. This procedure assumes that the database was created with the default block size for the specified IQ page size (or else the estimate is incorrect). Table 7-21 lists the sp_iqestdbspaces parameters.

Table 7-21: sp_iqestdbspaces parameters

Name	Datatype	Description
db_size_in_bytes	decimal(16)	Size of the database in bytes.
iq_page_size	smallint	The page size defined for the IQ segment of the database (must be a power of 2 between 65536 and 524288; the default is 131072).
min_#_of_bytes	int	The minimum number of bytes per dbspace segment. The default is 20,000,000 (20MB).
max_#_of_bytes	int	The maximum number of bytes per dbspace segment. The default is 2,146,304,000 (2.146GB).

Usage

sp_iqestdbspaces displays four types of recommendations, depending on how much of the data is unique:

min If there is little variation in data, you can choose to create only the dbspace segments of the sizes recommended as min. These recommendations reflect the best possible compression on data with the least possible variation.

avg If your data has an average amount of variation, create the dbspace segments recommended as min, plus additional segments of the sizes recommended as avg.

max If your data has a high degree of variation (many unique values), create the dbspace segments recommended as min, avg, and max.

spare If you are uncertain about the number of unique values in your data, create the dbspace segments recommended as min, avg, max, and spare. You can always delete unused segments after loading your data, but creating too few can cost you some time.

❖ Using sp_iqestdbspaces with other system stored procedures

1. Run sp_iqestjoin for all the table pairs you expect to join frequently.
2. Select one of the suggested index sizes for each pair of tables.
3. Total the index sizes you selected for all tables.
4. Run sp_iqestspace for all tables.
5. Total all of the RAW DATA index sizes returned by sp_iqestspace.
6. Add the total from step 3 to the total from step 5 to determine total index size.
7. Use the total index size calculated in step 6 as the *db_size_in_bytes* parameter in sp_iqestdbspaces.

Results of sp_iqestdbspaces are only estimates, based on the average size of an index. The actual size depends on the data stored in the tables, particularly on how much variation there is in the data.

Sybase strongly recommends that you create the spare dbspace segments, because you can delete them later if they are unused.

Example

```
sp_iqestdbspaces 12000000000, 65536, 500000000,
2146304000
```

dbspace files	Type	Size	Msg
1	min	2146304000	
2	min	2146304000	
3	min	507392000	
4	avg	2146304000	
5	max	2053697536	
6	spare	1200001024	

This example estimates the size and number of dbspace segments needed for a 12GB database. Sybase IQ recommends that you create a minimum of 3 segments (listed as min) for the best compression, if you expect little uniqueness in the data. If the data has an average amount of variation, 1 more segment (listed as avg) should be created. Data with a lot of variation (many unique values, requiring extensive indexing), may require 1 more segment (listed as max). You can ensure that your initial load succeeds by creating a spare segment of 1200001024 bytes. Once you have loaded the database, you can delete any unused dbspace segments.

sp_iqestspace procedure

Function

Estimates the amount of space needed to create an index based on the number of rows in the table.

Syntax

sp_iqestspace (*table_name, #_of_rows, iq_page_size*)

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Displays the amount of space that a database requires based on the number of rows in the underlying database tables and on the database IQ page size. This procedure assumes that the database was created with the default block size for the specified IQ page size (or else the estimate is incorrect). Table 7-22 lists the sp_iqestspace parameters.

Table 7-22: sp_iqestspace parameters

Name	Datatype	Description
table_name	char(256)	Name of the table
#_of_rows	int	Number of rows in the table
iq_page_size	smallint	The page size defined for the IQ segment of the database (must be a power of 2 between 65536 and 524288; the default is 131072)

sp_iqevent procedure

Function

Displays information about system and user-defined events.

Syntax

sp_iqevent [*event-name*], [*event-owner*], [*event-type*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

event-name The name of the event.

event-owner The owner of the event.

event-type The type of event. Allowed values are:

- SYSTEM: displays information about system events (events owned by user SYS or dbo) only
- ALL: displays information about user and system events
- Any other value: displays information about user events

The sp_iqevent procedure can be invoked without any parameters. If no parameters are specified, only information about user events (events not owned by dbo or SYS) is displayed by default.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqevent NULL, NULL, SYSTEM` and `sp_iqevent NULL, user1`.

Table 7-23: sp_iqevent usage examples

Syntax	Output
sp_iqevent	Displays information about all user events in the database
sp_iqevent e1	Displays information about the event e1
sp_iqevent non_existing_event	No rows returned, as the event non_existing_event does not exist
sp_iqevent NULL, DBA	Displays information about all events owned by DBA
sp_iqevent e1, DBA	Displays information about the event e1 owned by DBA
sp_iqevent ev_iqbegintxn	ev_iqbegintxn is a system-defined event. If there is no user-defined event also named ev_iqbegintxn, no rows are returned. (By default only user-defined events are returned.)
sp_iqevent ev_iqbegintxn, dbo	No rows returned, as the event ev_iqbegintxn is not a user event (by default only user events returned)
sp_iqevent NULL, NULL, SYSTEM	Displays information about all system events (owned by dbo or SYS)
sp_iqevent ev_iqbegintxn, NULL, SYSTEM	Displays information about the system event ev_iqbegintxn
sp_iqevent ev_iqbegintxn, dbo, ALL	Displays information about the system event ev_iqbegintxn owned by dbo

Description

The sp_iqevent stored event displays information about events in a database. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if *event-name* is specified, only information about the specified event is displayed. If *event-owner* is specified, sp_iqevent only returns information about events owned by the specified owner. If no parameters are specified, sp_iqevent displays information about all the user events in the database.

The sp_iqevent procedure returns information in the following columns:

Table 7-24: sp_iqevent columns

Column name	Description
event_name	The name of the event
event_owner	The owner of the event
event_type	For system events, the event type as listed in the SYSEVENTTYPE system table
enabled	Indicates whether or not the event is allowed to fire (Y/N)
action	The event handler definition
condition	The WHERE condition used to control firing of the event handler
location	The location where the event is allowed to fire: • C = consolidated • R = remote • A = all
remarks	A comment string

Examples

Display information about the user-defined event e1:

```
sp_iqevent e1

event_name    event_owner    event_type    enabled    action
e1            DBA           (NULL)         Y          (NULL)

condition    location    remarks
(NULL)       A           (NULL)
```

Display information about all system events:

```
sp_iqevent NULL, NULL, SYSTEM

event_name      event_owner    event_type      enabled   action
ev_iqbegintxn   dbo            IQTLVAvailable  Y         begin call
                                                         dbo.sp_iqlog...
ev_iqmpxcompact dbo           (NULL)           N         begin Declare
                                                         _Catalog...

condition    location    remarks
(NULL)       A           (NULL)
(NULL)       A           (NULL)
```

See also

CREATE EVENT statement in *Reference: Statements and Options*

Chapter 6, "Automating Tasks Using Schedules and Events" in the *System Administration Guide: Volume 2*

sp_iqfile procedure

Function

Displays detailed information about each dbfile in a dbspace.

Syntax

sp_iqfile [*dbspace-name*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

sp_iqfile displays the usage, properties, and types of data in each dbfile in a dbspace. You can use this information to determine whether data must be moved, and for data that has been moved, whether the old versions have been deallocated.

sp_iqfile displays the following information:

Table 7-25: sp_iqfile columns

Column name	Description
DBSpaceName	Name of the dbspace as specified in the CREATE DBSPACE statement. Dbspace names are case-insensitive for databases created with CASE RESPECT.
DBFileName	Logical file name.
Path	Location of the physical file or raw partition.
SegmentType	Type of dbspace (MAIN or TEMPORARY).
RWMode	Mode of the dbspace: read-write (RW) or read-only (RO).
Online	T (online) or F (offline).
Usage	Percent of dbspace currently in use by this file in the dbspace.
DBFileSize	Current size of the file or raw partition. For a raw partition, this size value can be less than the physical size.
Reserve	Reserved space that can be added to this file in the dbspace.
StripeSize	Amount of data written to the file before moving to the next file, if disk striping is on.
BlkTypes	Space used by both user data and internal system structures (see Table 7-26 for identifier values).
FirstBlk	First IQ block number assigned to the file.
LastBlk	Last IQ block number assigned to the file.
OkToDrop	‘Y’ indicates the file can be dropped; otherwise ‘N’.

Table 7-26 lists the values of the block type identifiers.

Table 7-26: sp_iqfile block types

Identifier	Block Type
A	Active Version
B	Backup Structures
C	Checkpoint Log
D	Database Identity
F	Free list
G	Global Free list Manager
H	Header Blocks of the Free List
I	Index Advice Storage
M	Multiplex CM*
O	Old Version
T	Table Use
U	Index Use
N	Column Use
X	Drop at Checkpoint

*The multiplex commit identity block (actually 128 blocks) exists in all IQ databases, even though it is not used by simplex databases.

Example

Displays information about the files in the dbspaces:

```
sp_iqfile;
```

```
DBSpace          DBFile           Path                                Segment      RWMode
 Name             Name                                                 Type
IQ_SYSTEM_MAIN  IQ_SYSTEM_MAIN  /sunopt/users/user1/iqdemo.iq      MAIN          RW
IQ_SYSTEM_TEMP  IQ_SYSTEM_TEMP  /sunopt/users/user1/iqdemo.iqtmp TEMPORARY     RW

Online Usage DBFileSize Reserve Stripesize BlkTypes  FirstBlk LastBlk OkToDrop
  T     21    300M       50M      8K       1H,7648F,    1      38400    N
                                           32D,128M
  T      1    100M       50M      8K       1H,64F,16A   1      12800    N
```

See also

“sp_iqdbspaceinfo procedure” on page 391, and “sp_iqindexinfo procedure” on page 421

Chapter 5, “Working with Database Objects,” in the *Sybase IQ System Administration Guide*

sp_iqhelp procedure

Function

Displays information about system and user-defined objects and data types.

Syntax

sp_iqhelp [*obj-name*], [*obj-owner*], [*obj-category*], [*obj-type*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

obj-name The name of the object.

obj-owner The owner of the object.

obj-category An optional parameter that specifies the category of the object.

Table 7-27: sp_iqhelp obj-category parameter values

object-type parameter	Specifies
"table"	The object is a base table
"view"	The object is an view
"procedure"	The object is a stored procedure or function
"event"	The object is an event
"datatype"	The object is a system or user-defined data type

Columns, constraints, and indexes are associated with tables and cannot be queried directly. When a table is queried, the information about columns, indexes, and constraints associated with that table is displayed.

If the specified object category is not one of the allowed values, an `"Invalid object category"` error is returned.

obj-type The type of object. Allowed values are:

- SYSTEM: displays information about system objects (objects owned by user SYS or dbo) only
- ALL: displays information about all objects

By default, only information about non-system objects is displayed. If the specified object type is not SYSTEM or ALL, an `"Invalid object type"` error is returned.

The sp_iqhelp procedure can be invoked without any parameters. If no parameters are specified, sp_iqhelp displays information about all independent objects in the database, that is, base tables, views, stored procedures, functions, events, and data types.

If you do not specify any of the first three parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqhelp NULL, NULL, NULL, SYSTEM` and `sp_iqhelp NULL, user1, "table"`.

Enclose the *obj-category* parameter in single or double quotes., except when NULL.

If sp_iqhelp does not find an object in the database that satisfies the specified description, the error `"No object found for the given description"` is returned.

Table 7-28: sp_iqhelp usage examples

Syntax	Output
sp_iqhelp	Displays summary information about all user-defined tables, views, procedures, events, and data types in the database
sp_iqhelp t1, u1, "table"	Displays information about table t1 owned by user u1 and the columns, indexes, and constraints associated with t1
sp_iqhelp NULL, u1, "view"	Displays information about view v1 owned by user u1 and the columns associated with v1
sp_iqhelp sp2	Displays information about the procedure sp2 and the parameters of sp2
sp_iqhelp e1	Displays information about the event e1
sp_iqhelp dt1	Displays information about the data type dt1
sp_iqhelp NULL, NULL, NULL, SYSTEM	Displays summary information about all system objects (owned by dbo or SYS)
sp_iqhelp non_existing_obj	Error `"Object 'non_existing_obj' not found"` returned, as the object non_existing_obj does not exist
sp_iqhelp NULL, non_existing_user	Error `"User 'non_existing_user' not found"` returned, as the user non_existing_user does not exist
sp_iqhelp t1, NULL, "apple"	Error `"Invalid object category 'apple'"` returned, as "apple" is not an allowed value for *obj-category*
sp_iqhelp t1, NULL, NULL, "USER"	Error `"Invalid object type 'USER'"` returned, as "USER" is not an allowed value for *obj-type*

Description

The sp_iqhelp stored procedure displays information about system and user-defined objects and data types in an IQ database. Objects supported by sp_iqhelp are tables, views, columns, indexes, join indexes, constraints, stored procedures, functions, events, and data types.

If you specify one or more parameters, the result is filtered by the specified parameters. For example, if *obj-name* is specified, only information about the specified object is displayed. If *obj-owner* is specified, sp_iqhelp returns information only about objects owned by the specified owner. If no parameters are specified, sp_iqhelp displays summary information about all user-defined tables, views, procedures, events, and data types in the database.

The sp_iqhelp procedure returns either summary or detailed information, depending on whether the specified parameters match multiple objects or a single object. The output columns of sp_iqhelp are similar to the columns displayed by the stored procedures sp_iqtable, sp_iqindex, sp_iqview, and sp_iqconstraint.

When multiple objects match the specified sp_iqhelp parameters, sp_iqhelp displays summary information about those objects.

Table 7-29: sp_iqhelp summary information

Object type	Columns displayed
base table	table_name, table_owner, server_type, location, table_constraints, remarks
view	view_name, view_creator, view_def, server_type, location, remarks
stored procedure	proc_name, proc_creator, proc_defn, replicate, srvid, remarks
function	proc_name, proc_creator, proc_defn, replicate, remarks
event	event_name, event_creator, enabled, location, event_type, action, external_action, condition, remarks
system and user-defined data types	type_name, creator, nulls, width, scale, default, check

When a single object matches the specified sp_iqhelp parameters, sp_iqhelp displays detailed information about the object.

Table 7-30: sp_iqhelp detailed information

Object type	Description	Columns
table	Displays information about the specified base table, its columns, indexes, constraints, and join indexes (if the table participates in any join indexes)	• Table columns: table_name, table_owner, server_type, location, table_constraints, remarks • Column columns: column_name, domain_name, width, scale, nulls, default, check, pkey, user_type, cardinality, est_cardinality, remarks • Index columns: index_name, column_name, index_type, unique_index, location, remarks • Constraint columns: constraint_name (role), column_name, index_name, constraint_type, foreigntable_name, foreigntable_owner, foreigncolumn_name, foreignindex_name, location • Join index columns: joinindex_name, creator, left_table_name, left_table_owner, left_column_name, join_type, right_table_name, right_table_owner, right_column_name, key_type, valid, remarks
view	Displays information about the specified view and its columns	• View columns: view_name, view_creator, view_def, server_type, location, remarks • Column columns: column_name, domain_name, width, scale, nulls, default, check, pkey, user_type, cardinality, est_cardinality, remarks
stored procedure	Displays information about the specified procedure and its parameters	• Procedure columns: proc_name, proc_creator, proc_defn, replicate, srvid, remarks • Parameter columns: parameter_name, type, width, scale, default, mode
function	Displays information about the specified function and its parameters	• Function columns: proc_name, proc_creator, proc_defn, replicate, srvid, remarks • Parameter columns: parameter_name, type, width, scale, default, mode
event	Displays information about the specified event	• Event columns: event_name, event_creator, enabled, location, event_type, action, external_action, condition, remarks
data type	Displays information about the specified data type	• Data type columns: type_name, creator, nulls, width, scale, default, check

For descriptions of the individual output columns listed in Table 7-30, refer to the descriptions of the following stored procedures:

- table: “sp_iqtable procedure” on page 472

- column: “sp_iqcolumn procedure” on page 370
- index: “sp_iqindex and sp_iqindex_alt procedures” on page 415
- constraint: “sp_iqconstraint procedure” on page 377
- join index: “sp_iqjoinindex procedure” on page 427
- view: “sp_iqview procedure” on page 486 and the Adaptive Server Enterprise catalog procedure sp_columns (for view columns)
- stored procedure and function: “sp_iqprocedure procedure” on page 443 and “sp_iqprocparm procedure” on page 446 (for procedure parameters)
- event: “sp_iqevent procedure” on page 404
- data type: “sp_iqdatatype procedure” on page 384

Adaptive Server Enterprise compatibility The Sybase IQ sp_iqhelp stored procedure is similar to the Adaptive Server Enterprise sp_help procedure, which displays information about any database object listed in the SYSOBJECTS system table and about system and user-defined data types.

Sybase IQ has some architectural differences from Adaptive Server in terms of types of objects supported and the namespace of objects. In Adaptive Server, all objects (tables, views, stored procedures, logs, rules, defaults, triggers, check constraints, referential constraints, and temporary objects) are stored in the SYSOBJECTS system table and are in the same namespace. The objects supported by Sybase IQ (tables, views, stored procedures, events, primary keys, and unique, check, and referential constraints) are stored in different system tables and are in different namespaces. For example, in Sybase IQ a table can have the same name as an event or a stored procedure.

Because of the architectural differences between Sybase IQ and Adaptive Server, the types of objects supported by and the syntax of Sybase IQ sp_iqhelp are different from the supported objects and syntax of Adaptive Server sp_help; however, the type of information about database objects that is displayed by both stored procedures is similar.

Examples Display detailed information about the table sale:

```
sp_iqhelp sale
```

```
 Table_name Table_owner Server_type Location dbspace_id isPartitioned
 table_constraints
========== =========== ==========  =======  ========= =============

 sale      DBA         IQ          Main    16387      N

Remarks  table_constraints
=======  ==================

(NULL)   (NULL)

column_name domain_name width scale nulls default cardinality
==========  =========== ===== ===== ===== ======= ===========
prod_id     integer     4     0     Y    (NULL)   0
month_num   integer     4     0     Y    (NULL)   0
rep_id      integer     4     0     Y    (NULL)   0
sales       integer     4     0     Y    (NULL)   0

  est_cardinality    isPartitioned    remarks    check
  ==============     =============    =======    =====
  0                  N               (NULL)      (NULL)
  0                  N               (NULL)      (NULL)
  0                  N               (NULL)      (NULL)
  0                  N               (NULL)      (NULL)

index_name           column_name  index_type  unique_index  location
==========           ===========  =========== ===========   ========
ASIQ_IDX_T463_C2_FP  month_num    FP          N             Main
ASIQ_IDX_T463_C1_FP  prod_id      FP          N             Main
ASIQ_IDX_T463_C3_FP  rep_id       FP          N             Main
ASIQ_IDX_T463_C4_FP  sales        FP          N             Main

  remarks
  =======
  (NULL)
  (NULL)
  (NULL)
  (NULL)
```

Display detailed information about the procedure sp_customer_list:

```
sp_iqhelp sp_customer_list
```

```
proc_name    proc_owner    proc_defn
==========  ===========    =========
sp_customer_list    DBA    create procedure DBA.sp_customer_list()
                           result(id integer company_name char(35))
                           begin
                           select id company_name from Customers
                           end

  replicate    srvid    remarks
  =========    =====    =======
  N            (NULL)   (NULL)

parm_name      parm_type  parm_mode  domain_name  width  scale
=========      =========  =========  ===========  =====  =====
id             result     out        integer      4      0
company_name   result     out        char         35     0

  default
  =======
  (NULL)
```

sp_iqindex and sp_iqindex_alt procedures

Function

Lists information about indexes.

Syntax 1

sp_iqindex ([*table_name*],[*column_name*],[*table_owner*])

Syntax 2

sp_iqindex [table_name=*'tablename'*],
[column_name=*'columnname'*],[table_owner=*'tableowner'*]

Syntax 3

sp_iqindex_alt ([*table_name*],[*column_name*],[*table_owner*])

Syntax 4

sp_iqindex_alt [table_name=*'tablename'*],
[column_name=*'columnname'*],[table_owner=*'tableowner'*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

Syntax1 If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqindex NULL,NULL,DBA` and `sp_iqindex Departments,NULL,DBA`.

Syntax2 You can specify the parameters in any order. Enclose them in single quotes.

Syntax 3 and 4 Produces slightly different output when a multicolumn index is present. Allows the same options as Syntax 1 and 2.

Description

Displays information about indexes in the database. Specifying one of the parameters returns the indexes from only that table, column, or tables owned by the specified user. Specifying more than one parameter filters the results by all of the parameters specified. Specifying no parameters returns all indexes for all tables in the database.

Table 7-31: sp_iqindex and sp_iqindex_alt columns

Column name	Description
table_name	The name of the table
table_owner	The owner of the table
column_name	The name of the column; multiple names can appear in a multicolumn index
index_type	The abbreviated index type (for example, HG, LF)
index_name	The name of the index
unique_index	'U' indicates the index is a unique index; otherwise, 'N'
location	TEMP = IQ temporary store, MAIN = IQ store, SYSTEM = catalog store
remarks	User comments added with the COMMENT statement

sp_iqindex always produces one line per index. sp_iqindex_alt produces one line per index per column if there is a multicolumn index.

Examples

The following variations in syntax both return all indexes on columns with the name DepartmentID:

```
call sp_iqindex (NULL,'DepartmentID')
sp_iqindex column_name='DepartmentID'
```

table_name	table_owner	column_name	index_type	index_name	unique_index	location	dbspace_id	remarks
Departments	GROUPO	DepartmentID	FP	ASIQ_IDX_T201_C1_FP	N	Main	16387	(NULL)
Departments	GROUPO	DepartmentID	HG	ASIQ_IDX_T201_C1_HG	U	Main	16387	(NULL)
Employees	GROUPO	DepartmentID	FP	ASIQ_IDX_T202_C5_FP	N	Main	16387	(NULL)

The following variations in syntax both return all indexes in the table Departments that is owned by table owner GROUPO:

```
sp_iqindex Departments,NULL,GROUPO
sp_iqindex table_name='Departments',table_owner='DBA'
```

table_name	table_owner	column_name	index_type	index_name	unique_index	location	dbspace_id	remarks
Departments	GROUPO	DepartmentHeadID	FP	ASIQ_IDX_T201_C3_FP	N	Main	16387	(NULL)
Departments	GROUPO	DepartmentID	FP	ASIQ_IDX_T201_C1_FP	N	Main	16387	(NULL)
Departments	GROUPO	DepartmentID	HG	ASIQ_IDX_T201_C1_HG	U	Main	16387	(NULL)
Departments	GROUPO	DepartmentName	FP	ASIQ_IDX_T201_C2_FP	N	Main	16387	(NULL)

The following variations in syntax for sp_iqindex_alt both return indexes on the table Employees that contain the column City. The index emp_loc is a multicolumn index on the columns City and State. sp_iqindex_alt displays one row per column for a multicolumn index.

```
sp_iqindex_alt Employees,City
sp_iqindex_alt table_name='Employees',
               column_name='City'
```

table_name	table_owner	column_name	index_type	index_name	unique_index	dbspace_id	remarks
Employees	GROUPO	City	FP	ASIQ_IDX_T452_C7_FP	N	16387	(NULL)
Employees	GROUPO	City	HG	emp_loc	N	16387	(NULL)
Employees	GROUPO	State	HG	emp_loc	N	16387	(NULL)

The output from sp_iqindex for the same table and column is slightly different:

```
sp_iqindex Employees,City
sp_iqindex table_name='Employee',column_name='City'
```

table_name	table_owner	column_name	index_type	index_name	unique_index	dbspace_id	location	remarks
Employees	GROUPO	City	FP	ASIQ_IDX_T452_C7_FP	N	16387	Main	(NULL)

table_ name	table_ owne r	column_ name	index_ type	index_name	unique_ index	dbspa ce_id	locatio n	remark s
Employe es	GRO UPO	City,State	HG	emp_loc	N	16387	Main	(NULL)

See also

"FP_LOOKUP_SIZE option," "INDEX_ADVISOR option," and "MINIMIZE_STORAGE option" in Chapter 2, "Database Options" in *Reference: Statements and Options*

Chapter 6, "Using Sybase IQ Indexes" in *System Administration Guide: Volume 1*.

sp_iqindexadvice procedure

Function

Displays stored index advice messages. Optionally clears advice storage.

Syntax

sp_iqindexadvice ([*resetflag*])

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

resetflag Lets the caller clear the index advice storage. If *resetflag* is nonzero, all advice is removed after the last row has been retrieved.

Description

Allows users to query aggregated index advisor messages using SQL. Information can be used to help decide which indexes or schema changes will affect the most queries.

Table 7-32 lists INDEX_ADVISOR columns.

Table 7-32: sp_iqindexadvice columns

Column name	Description
Advice	Unique advice message
NInst	Number of instances of message
LastDT	Last date/time advice was generated

Examples

Table 7-33 illustrates sample output from the sp_iqindexadvice procedure.

Table 7-33: Sample sp_iqindexadvice output

Advice	NInst	LastDT
Add a CMP index on DBA.tb (c2, c3) Predicate: (tb.c2 = tb.c3)	2073	2009-04-07 16:37:31.000
Convert HG index on DBA.tb.c4 to a unique HG	812	2009-04-06 10:01:15.000
Join Key Columns DBA.ta.c1 and DBA.tb.c1 have mismatched data types	911	2009-02-25 20:59:01.000

See also

"sp_iqcolumnuse procedure" on page 372, "sp_iqdbspace procedure" on page 388, "sp_iqindexuse procedure" on page 426, "sp_iqtableuse procedure" on page 477, "sp_iqunusedcolumn procedure" on page 481, "sp_iqunusedindex procedure" on page 482, "sp_iqunusedtable procedure" on page 483, and "sp_iqworkmon procedure" on page 490

"FP_LOOKUP_SIZE option," "INDEX_ADVISOR option," and "MINIMIZE_STORAGE option" in Chapter 2, "Database Options" in *Reference: Statements and Options*

Chapter 6, "Using Sybase IQ Indexes" in *System Administration Guide: Volume 1*.

sp_iqindexfragmentation procedure

Function

Reports information about the percentage of page space taken up within the B-trees, garrays, and bitmap structures in Sybase IQ indexes.

For garrays, the fill percentage calculation does not take into account the reserved space within the garray groups, which is controlled by the GARRAY_FILL_FACTOR_PERCENT option.

Syntax

dbo.sp_iqindexfragmentation ('*target* ')

target: **table** *table-name* | index *index-name* [...]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

table-name Target table *table-name* reports on all nondefault indexes in the named table.

index-name Target index *index-name* reports on the named index. Each index-name is a qualified index name. You can specify multiple indexes within the table, but you must repeat the index keyword with each index specified.

Example

Reports the internal index fragmentation for nonunique HG index cidhg in table Customers:

```
dbo.sp_iqindexfragmentation ('index customers.cidhg')
```

Index	Index type	btree node pages	GARRAY_FILL_FACTOR_PERCENT
dba.customers.cidhg	HG	3	75
SQLCODE	0		
Fill Percent	**btree pages**	**garray pages**	**bitmap pages**
0 - 10%	0	0	0
11 - 20%	0	0	0
21 - 30%	0	0	0
31-40%	0	0	22
41 - 50%	0	0	0
51 - 60%	0	0	10
61 - 70%	2	0	120
71 - 80%	138	3	64
81 - 90%	24	122	14
91 - 100%	18	1	0

According to this output, of the 182 B-tree pages in nonunique HG index cidhg, 2 are between 61% and 70% full, 138 are 71% to 80% full, 24 are 81% - 90% full, and 18 are 91% - 100% full. Usage for garray and bitmap pages is reported in the same manner. All percentages are truncated to the nearest percentage point. HG indexes also display the value of option GARRAY_FILL_FACTOR_PERCENT. Those index types that use a B-tree also display the number of node (nonleaf) pages. These are HG, LF, WD, DATE, and DTTM.

If an error occurred during execution of the stored procedure for this index, the SQLCODE would be nonzero.

See also

"GARRAY_FILL_FACTOR_PERCENT option" and "GARRAY_PAGE_SPLIT_PAD_PERCENT option," Chapter 2, "Database Options," in *Reference: Building Blocks, Tables, and Procedures.*

"FP_LOOKUP_SIZE option," "INDEX_ADVISOR option," and "MINIMIZE_STORAGE option" in Chapter 2, "Database Options" in *Reference: Statements and Options*

Chapter 6, "Using Sybase IQ Indexes" in *System Administration Guide: Volume 1*.

sp_iqindexinfo procedure

Function

Displays the number of blocks used per index per main dbspace for a given object. If the object resides on several dbspaces, sp_iqindexinfo returns the space used in all dbspaces, as shown in the example.

Syntax

sp_iqindexinfo '{ **database**
| [**table** *table-name* | **index** *index-name*] [...] }
[**resources** *resource-percent*]'

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

You can request index information for the entire database, or you can specify any number of table or index parameters. If a table name is specified, sp_iqindexinfo returns information on all indexes in the table. If an index name is specified, only the information on that index is returned.

You cannot specify a join index by name. Use the database keyword to display join indexes.

If the specified *table-name* or *index-name* is ambiguous or the object cannot be found, an error is returned.

By default in a multiplex database, sp_iqindexinfo displays information about the shared IQ store on a secondary node. If individual tables or indexes are specified, the store to display is automatically selected .

resource-percent must be an integer greater than 0. The resources percentage allows you to limit the CPU utilization of the sp_iqindexinfo procedure by specifying the percent of total CPUs to use.

Description

sp_iqindexinfo shows the DBA on which dbspaces a given object resides. The DBA can use this information to determine which dbspaces must be given relocate mode to relocate the object.

The results of sp_iqindexinfo are from the point of view of the version seen by the transaction running the command. Blocks used by other versions are not shown.

Table 7-34: sp_iqindexinfo columns

Column name	Description
Object	Table, index, or join index name
Dbspace_name	Name of the dbspace
ObjSize	Size of data for this object on this dbspace
DBSpPct	Percent of dbspace used by this object
MinBlk	First block used by this object on this dbspace

Column name	Description
MaxBlk	Last block used by this object on this dbspace; useful for determining which objects must be relocated before the dbspace is resized to a smaller size

Examples

Displays index information about the table t2:

```
sp_iqindexinfo 'table t2';
```

Object	dbspace_name	ObjSize	DBSpPct	MinBlk	MaxBlk
t2	IQ_SYSTEM_MAIN	32K	1	84	107
t2	dbspacedb2	160K	2	1045495	1045556
t2	dbspacedb3	8K	1	2090930	2090930
t2.DBA.ASIQ_IDX_T430_C1_FP	IQ_SYSTEM_MAIN	136K	2	126	321
t2.DBA.ASIQ_IDX_T430_C1_FP	dbspacedb3	152K	2	2091032	2091053
t2.DBA.t2c1hng	dbspacedb2	136K	2	1045537	1045553

See also

“sp_iqdbspace procedure” on page 388, “sp_iqdbspaceinfo procedure” on page 391, “sp_iqspaceinfo procedure” on page 459.

Chapter 5, “Working with Database Objects” in the *System Administration Guide: Volume 1*

Appendix A, “Multiplex Reference” in *Using Sybase IQ Multiplex*.

sp_iqindexmetadata procedure

Function

Displays index metadata for a given index. You can optionally restrict the output to only those indexes on a specified table, and to only those indexes belonging to a specified owner.

Syntax

dbo.sp_iqindexmetadata {'*index-name*'
[, '*table-name*' [, '*owner-name*']] }

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

Specifying a table name limits output to those indexes belonging to that table. Specifying an owner name limits output to indexes owned by that owner. Omitted parameters default to NULL. You can specify only one index per procedure.

Description

The first row of output is the owner name, table name, and index name for the index.

Subsequent rows of output are specific to the type of index specified.

Table 7-35: sp_iqindexmetadata output rows

Index type	Metadata returned
CMP, DATE, DTTM, TIME	Type, Version
FP	Type, Version, LookupPages, Style, LookupEntries, 1stLookupPage, LargeLOBs, SmallLOBs, IQ Unique, LOB Compression (only if column datatype is LONG VARCHAR or LONG BINARY)
HG	Type, Version, Distinct Keys
HNG	Type, Version, BitsPerBlockmap, NumberOfBits
LD	Type, Version<ld>, Version, Distinct Keys
LF	Type, Version, IndexStatus, NumberOfBlockmaps, BitsPerBlockmap, Distinct Keys
WD	Type, Version, KeySize, Delimiters, DelimiterCount, MaxKeyWordLength, PermitEmptyWord

Examples

The following command displays index information about the HG index hg_index_col54:

```
sp_iqindexmetadata 'hg_index_col54' , 'metal' , 'DBA';
```

'DBA',	'metal'	'hg_index_col54'
'Type',	'HG',	''
'Version',	'2',	''
'Distinct Keys',	'0',	''

See also

“sp_iqindex and sp_iqindex_alt procedures” on page 415, “sp_iqindexfragmentation procedure” on page 419, “sp_iqindexinfo procedure” on page 421, and “sp_iqindexsize procedure” on page 425

Chapter 5, “Working with Database Objects” and Chapter 6, “Using Sybase IQ Indexes” in *System Administration Guide: Volume 1* in the *System Administration Guide: Volume 1*

“FP_LOOKUP_SIZE option” and “MINIMIZE_STORAGE option” in Chapter 2, “Database Options” in *Reference: Statements and Options*.

Chapter 6, “Using Sybase IQ Indexes” in *System Administration Guide: Volume 1*.

sp_iqindexsize procedure

Function

Gives the size of the specified index.

Syntax

sp_iqindexsize [[*owner*.] *table*.] *index_name*

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Table 7-36: sp_iqindexsize columns

Column name	Description
Username	Index owner.
Indexname	Index for which results are returned, including the table name.
Type	Index type.
Info	Component of the IQ index for which the KBytes, Pages, and Compressed Pages are being reported. The components vary by index type. For example, the default (FP) index includes BARRAY (barray) and Bitmap (bm) components. The Low_Fast (LF) index includes B-tree (bt) and Bitmap (bm) components.
KBytes	Physical object size in KB.
Pages	Number of IQ pages needed to hold the object in memory.
Compressed Pages	Number of IQ pages when the object is compressed (on disk).

Returns the total size of the index in bytes and kilobytes, and an Info column that describes the component of the IQ index for which the KBytes, Pages, and Compressed Pages are reported. The components described vary by index type. For example, the default (FP) index includes BARRAY (barray) and Bitmap (bm) components. The Low_Fast (LF) index includes B-tree (bt) and Bitmap (bm) components.

Also returns the number of pages required to hold the object in memory and the number of IQ pages when the index is compressed (on disk).

You must specify the *index_name* parameter with this procedure. To restrict results to this index name in a single table, include *owner.table.* when specifying the index.

Example

```
sp_iqindexsize ASIQ_IDX_T452_C19_FP
```

Username	Indexname	Type	Info	KBytes	Pages	Compressed Pages
DBA	Employees.ASIQ_IDX_T452_C19_FP	FP	Total	288	4	2

Username	Indexname	Type	Info	KBytes	Pages	Compressed Pages
DBA	Employees.ASIQ_IDX_T452_C19_FP	FP	vdo	0	0	0
DBA	Employees.ASIQ_IDX_T452_C19_FP	FP	bt	0	0	0
DBA	Employees.ASIQ_IDX_T452_C19_FP	FP	garray	0	0	0
DBA	Employees.ASIQ_IDX_T452_C19_FP	FP	bm	136	2	1
DBA	Employees.ASIQ_IDX_T452_C19_FP	FP	barray	152	2	1
DBA	Employees.ASIQ_IDX_T452_C19_FP	FP	dpstore	0	0	0
DBA	Employees.ASIQ_IDX_T452_C19_FP	FP	largelob	0	0	0

See also

Chapter 6, “Using Sybase IQ Indexes” in *System Administration Guide: Volume 1*.

“FP_LOOKUP_SIZE option” and “MINIMIZE_STORAGE option” in Chapter 2, “Database Options” in *Reference: Statements and Options*.

sp_iqindexuse procedure

Function

Reports detailed usage information for secondary (non-FP) indexes accessed by the workload.

Syntax

sp_iqindexuse

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Each secondary index accessed by the workload displays a row. Indexes that have not been accessed do not appear. Index usage is broken down by optimizer, constraint, and query usage.

Indexes from tables created in SYSTEM are not reported.

Table 7-37: sp_iqindexuse columns

Column name	Description
IndexName	Index name
TableName	Table name
Owner	User name of index owner
UID**	Index unique identifier
Type	Index type
LastDT	Date/time of last access
NOpt	Number of metadata/uniqueness accesses
NQry	Number of query accesses
NConstraint	Number of accesses for unique or referential integrity checks

**UID is a number assigned by the system that uniquely identifies the instance of the index (where instance is defined when an object is created).

Example

Sample output from the sp_iqindexuse procedure.

```
IndexName         TableName  Owner UID Type LastDT                NOpt  NQry  NConstraint
n_nationkey_hg    nation     DBA   29  HG   20070917 22:08:06~    12    0     12
n_regionkey_hg    nation     DBA   31  HG   20070917 22:08:06~    12    0      0
r_regionkey_hg    region     DBA   47  HG   20070917 22:08:06~    12    0     12
s_suppkey_hg      supplier   DBA   64  HG   20070917 22:08:06~    12    0     12
p_partkey_hg      part       DBA   87  HG   20070917 22:08:06~     6    0      6
s_suppkey_hg      supplier   DBA   64  HG   20070917 22:08:06~    12    0     12
...
```

See also

“Monitoring workloads,”Chapter 3, “Optimizing Queries and Deletions,” in the *Performance and Tuning Guide*.

“sp_iqcolumnuse procedure” on page 372, “sp_iqdbspace procedure” on page 388, “sp_iqindexadvice procedure” on page 418, “sp_iqtableuse procedure” on page 477, “sp_iqunusedcolumn procedure” on page 481, “sp_iqunusedindex procedure” on page 482, “sp_iqunusedtable procedure” on page 483, and “sp_iqworkmon procedure” on page 490

“INDEX_ADVISOR option” in Chapter 2, “Database Options,” in *Reference: Statements and Options*

sp_iqjoinindex procedure

Function

Displays information about join indexes.

Syntax

sp_iqjoinindex [*left-table-name*], [*left-column-name*], [*left-table-owner*], [*right-table-name*], [*right-column-name*], [*right-table-owner*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

left-table-name The name of the table that forms the left side of the join operation.

left-column-name The name of the column that is part of the left side of the join.

left-table-owner The owner of the table that forms the left side of the join operation.

right-table-name The name of the table that forms the right side of the join operation.

right-column-name The name of the column that is part of the right side of the join.

right-table-owner The owner of the table that forms the right side of the join operation.

The sp_iqjoinindex procedure can be invoked without any parameters. If no parameters are specified, sp_iqjoinindex displays information about all join indexes on IQ base tables. Note that join index tables are always IQ base tables. Join index tables cannot be temporary tables, remote tables, or proxy tables.

If you do not specify any of the first five parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqjoinindex NULL, NULL, NULL, t2, n2, DB'` and `sp_iqjoinindex t1, NULL, NULL, t2`.

Table 7-38: sp_iqjoinindex usage examples

Syntax	Output
sp_iqjoinindex	Displays information about all the join indexes
sp_iqjoinindex t1, NULL, DBA	Displays information about all join indexes in which t1 owned by DBA forms the left side of the operation
sp_iqjoinindex t2, n1, DBA	Displays join index information with column n1 of table t2 owned by DBA as left side of the join
sp_iqjoinindex NULL, NULL, DBA, NULL, NULL, DBA	Displays information about all join indexes in which the left and right side tables are owned by DBA
sp_iqjoinindex NULL, NULL, NULL, t2, NULL, NULL	Displays information about all join indexes in which the table t2 is on the right side of the join operation
sp_iqjoinindex t1, n1, DBA, t2, n1, DBA	Displays information about join indexes in which the left side is column n1 of table t1 owned by DBA and the right side is column n1 of table t2 owned by DBA
sp_iqjoinindex non_existing_table	No rows returned, as the table non_existing_table does not exist
sp_iqjoinindex NULL, NULL, non_existing_user	No rows returned, as the user non_existing_user does not exist

See also

CREATE JOIN INDEX statement in *Reference: Statements and Options*

Chapter 6, "Using Sybase IQ Indexes" in the *System Administration Guide: Volume 1*

Description

The sp_iqjoinindex stored procedure displays information about join indexes in a database. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if *left-table-name* is specified, sp_iqjoinindex displays all the join indexes in which that table forms the left side of the join. If *left-table-owner* is specified, sp_iqjoinindex only returns join indexes in which the left table is owned by the specified owner. If no parameters are specified, sp_iqjoinindex displays information about all join indexes in the database.

The sp_iqjoinindex procedure returns information in the following columns:

Table 7-39: sp_iqjoinindex columns

Column name	Description
joinindex_name	The name of the join index.
creator	The owner of the join index.
left_table_name	The name of the table that forms the left side of the join operation.
left_table_owner	The name of the owner of the table that forms the left side of the join operation.
left_column_name	The name of the column that is part of the left side of the join.
join_type	The only currently supported value is "=".
right_table_name	The name of the table that forms the right side of the join operation.
right_table_owner	The name of the owner of the table that forms the right side of the join operation.
right_column_name	The name of the column that is part of the right side of the join.
key_type	Defines the type of join on the keys: • NATURAL: a natural join • KEY: a key join • ON: a left outer/right outer/full join
valid	Indicates whether this join index needs to be synchronized. 'Y' means that it does not require synchronization; 'N' means that it does require synchronization.
remarks	A comment string.
dbspace_id	Name of the dbspace in which specified join indexes reside.

Examples

Displays information about the join index in which table t1 forms the left side of the join operation:

```
sp_iqjoinindex t1

joinnidex_name creator left_table_name left_table_owner left_column_name
join_type right_table_name right_table_owner right_column_name key_type
valid dbspace_id remarks
t1_t2_t3_join  DBA     t1              DBA               n1
=         t2                DBA                n1                 NATURAL
Y    16387    (NULL)
```

Displays information about the join index in which table t2 forms the left side of the join operation:

```
sp_iqjoinindex t2
```

```
joinnidex_name creator left_table_name left_table_owner left_column_name
join_type right_table_name right_table_owner right_column_name key_type
valid dbspace_id remarks
t1_t2_t3_join  DBA     t2              DBA              n1
=        t3              DBA              n1               NATURAL
Y    (NULL)
t1_t2_t3_join  DBA     t2              DBA              name
=        t3              DBA              name             NATURAL
Y    16387    ((NULL)
```

Displays information about join indexes in which the left side is column name of table t2 owned by DBA and the right side is column name of table t3 owned by DBA:

```
sp_iqjoinindex t2, name, DBA, t3, name, DBA

joinindex_name creator left_table_name left_table_owner left_column_name
join_type right_table_name right_table_owner right_column_name key_type
valid dbspace_id remarks
t1_t2_t3_join  DBA     t2              DBA              name
=        t3              DBA              name             NATURAL
Y    16387    ((NULL)
```

sp_iqjoinindexsize procedure

Function

Gives the size of the specified join index.

Syntax

sp_iqjoinindexsize (*join_index_name*)

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Returns the total size of the index in bytes, KBytes,and NBlocks (IQ blocks). Also returns the number of pages required to hold the join index in memory and the number of IQ pages when the join index is compressed (on disk). You must specify the *join_index_name* parameter with this procedure.

Table 7-40: sp_iqjoinindexsize columns

Column name	Description
Username	Owner of the join index
JoinIndexName	Join index for which results are returned
Number of Tables	Number of tables in the join index
KBytes	Physical object size in KB
Pages	Number of IQ pages needed to hold the object in memory
Compressed Pages	Number of IQ pages when the object is compressed (on disk)
NBlocks	Number of IQ blocks

Example

```
sp_iqjoinindexsize ( 't1t2' )
```

Username	JoinIndexName	Number of Tables	KBytes	Pages	Compressed Pages	NBlocks
DBA	t1t2	2	13	15	4	26

sp_iqlmconfig procedure

Function

Controls license management configurations, displays and sets license type and status.

Syntax 1

sp_iqlmconfig ’edition’, { ’SE’ | ’SA’ | ’EE’ }

Table 7-41: Summary information for "edition" parameter

Topic	Value
Default	’EE’ (Enterprise Edition)
Range of values	’SE’ (Small Business) ’SA’ (Single Application) ’EE’ (Enterprise Edition)
Status	Static

Syntax 2

sp_iqlmconfig ’license type’, { ’CP’ | ’DT’ | ’SF’ | ’AC’ | ’BC’ | 'CH' | 'DH' | 'SH' | 'AH' | 'BH' }

Table 7-42: Summary information for "license type" parameter

Topic	Value
Default	’DT’ (Development and Testing)
Range of values	’AC’ (OEM CPU License) ’AH’ (OEM CPU License Chip) ’BC’ (OEM Standby License) ’BH’ (OEM Standby License Chip) ’CP’ (CPU License) ’CH’ (CPU License Chip) ’DH’ (Development and Testing License Chip) ’DT’ (Development and Testing) ’EV’ (Evaluation) ’SF’ (Standby CPU License) ’SH’ (Standby CPU License Chip)
Status	Static

Syntax 3

```
sp_iqlmconfig ’email severity’, { ’ERROR’ | ’WARNING’ |
’INFORMATIONAL’ | ’NONE’ }
```

NONE designates that e-mail notification is disabled.

Syntax 4

```
sp_iqlmconfig ’smtp host’, ’<hostname>’ | ’
```

hostname Specifies SMTP host used for e-mail notification.

Syntax 5

```
sp_iqlmconfig ’email sender’, ’<email address>’ |
```

<email address> Specifies the sender’s e-mail address used for e-mail notification.

Syntax 6

```
sp_iqlmconfig ’email recipients’, ’<email recipients>’ |
```

<email recipients> Specifies a comma-separated list of e-mail addresses to whom e-mail notifications will be sent.

Syntax 7

```
sp_iqlmconfig |
```

Permissions

DBA permissions necessary.

Usage

At startup, sp_iqlmconfig checks the edition type and license type of the license specified.

- If the specified license is not found, the server falls to grace mode.
- The specified license type becomes valid only when a non-null edition value is specified.
- If sp_iqlmconfig is called with no parameters (Syntax 3), it displays all the information above, as well as other information, such as:

- Product Edition and License Type
- Which of the optional licenses are in use
- License count
- E-mail information
- General information about the license

sp_iqlocks procedure

Function

Shows information about locks in the database, for both the IQ store and the catalog store.

Syntax

sp_iqlocks ([*connection*,] [[*owner*.]*table_name*] *max_locks*,] [*sort_order*])

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

Table 7-43 lists the optional sp_iqlocks parameters you can specify to restrict results.

Table 7-43: Optional sp_iqlocks parameters

Name	Data type	Description
connection	integer	Connection ID. With this option, the procedure returns information about locks for the specified connection only. Default is zero, which returns information about all connections.
owner.table_ name	char (128)	Table name. With this option, the procedure returns information about locks for the specified table only. Default is NULL, which returns information about all tables in the database. If you do not specify *owner*, it is assumed that the caller of the procedure owns the table.
max_locks	integer	Maximum number of locks for which to return information. Default is 0, which returns all lock information.
sort_order	char(1)	Order in which to return information: • C sorts by connection (default) • T sorts by table_name

Description

Displays information about current locks in the database. Depending on the options you specify, you can restrict results to show locks for a single connection, a single table, or a specified number of locks.

sp_iqlocks displays the following information, sorted as specified in the *sort_order* parameter:

Table 7-44: sp_iqlocks columns

Column	Description
conn_name	The name of the current connection.
conn_id	Connection ID that has the lock.
user_id	User associated with this connection ID.
table_type	The type of table. This type is either BASE for a table, GLBTMP for global temporary table, or MVIEW for a materialized view.
creator	The owner of the table.
table_name	Table on which the lock is held.
index_id	The index ID or NULL
lock_class	String of characters indicating the type of lock: S – share. SW – share and write. EW – exclusive and write. E – exclusive. P – phantom. A – antiphantom. W – write. All locks listed have one of S, E, EW, or SW, and may also have P, A, or both. Phantom and antiphantom locks also have a qualifier of T or *: T – the lock is with respect to a sequential scan. * – the lock is with respect to all scans. *nnn* – Index number; the lock is with respect to a particular index. Sybase IQ obtains a share lock before a write lock. If a connection has exclusive lock, share lock does not appear. For write locks, if a connection has all-exclusive, share, and write locks, it is EW.
lock_type	Value identifying the lock (dependent on the lock class)
row_identifier	The identifier for the row or NULL.

If sp_iqlocks cannot find the connection ID or user name of the user who has a lock on a table, it displays a 0 (zero) for the connection ID and `User unavailable` for the user name.

Note Exclusive, phantom, or antiphantom locks can be placed on SQL Anywhere tables, but not on Sybase IQ tables. Unless you have explicitly taken out locks on a table in the catalog store, you never see these types of locks (or their qualifiers T, *, and *nnn*) in a Sybase IQ database. For information on how locking works in SQL Anywhere tables, see *SQL Anywhere Server – SQL Usage*.

Examples

The example shows the sp_iqlocks procedure call and its output in a Sybase IQ database. The procedure is called with all default options, so that the output shows all locks, sorted by connection.

```
call sp_iqlocks()

conn_name  conn_id  user_id  table_type  creator  table_name
=========  =======  =======  ==========  =======  ==========
con1       70187172 'mary'   BASE        DBA       t1

index_id              lock_class  lock_duration  lock_type  row_identifier
========              ==========  =============  =========  ==============
ASIQ_IDX_T452_C19_FP  Table       Position       Table      1
```

sp_iqmodifyadmin procedure

Function

Sets an option on a named login policy to a certain value. If no login policy is specified, the option is set on the root policy. In a multiplex, sp_iqmodifyadmin takes an optional parameter that is the multiplex server name.

Syntax1

call sp_iqmodifyadmin (‘*policy_option_name*’,
‘*value_in*’ ,[‘*login_policy_name*’])

Syntax2

sp_iqmodifyadmin ‘*policy_option_name*’,
‘*value_in’* ,’*login_policy_name* ’

Syntax3

sp_iqmodifyadmin *policy_option_name*, *value_in*, ,*login_policy_name*

Syntax 4

sp_iqmodifyadmin ‘*policy_option_name*’,
‘*value_in’* ,’*login_policy_name* ’ ,’*server_name* ’

Usage

policy_option_name The login policy option to be changed.

value_in New value for the login policy option.

login_policy_name Name of the login policy whose login policy option needs to be changed.

Permissions

Requires DBA authority.

See also

“sp_iqpassword procedure” on page 441

ALTER LOGIN POLICY statement in Chapter 1, “SQL Statements,” in *Reference: Statements and Options*.

Examples

Example 1 Sets the login option locked to ON for the policy named *lockeduser*.

```
call sp_iqmodifyadmin ('locked','on','lockeduser')
```

Example 2 Sets the login option locked to ON for the policy named *lockeduser* on the multiplex server named Writer1.

```
call sp_iqmodifyadmin
('locked','on','lockeduser','Writer1')
```

sp_iqmodifylogin procedure

Function

Assigns a user to a login policy.

Syntax1

call sp_iqmodifylogin '*userid*', ['*login_policy_name*']

Syntax2

sp_iqmodifylogin '*userid*', ['*login_policy_name*']

Permissions

DBA authority required.

Usage

userid Variable that holds the name of the account to modify.

login_policy_name (Optional) Specifies the name of the login policy to which the user will be assigned. If no login policy name is specified, the user is assigned to the root login policy.

Examples

Example 1 Assigns user joe to a login policy named expired_password:

```
sp_iqmodifylogin 'joe', 'expired_password'
```

Example 2 Assigns user joe to the root login policy:

```
call sp_iqmodifylogin ('joe')
```

See also

“sp_iqmodifyadmin procedure” on page 437

sp_iqobjectinfo procedure

Function

Returns partitions and dbspace assignments of database objects and sub-objects.

Syntax

sp_iqobjectinfo [*owner_name*] [, *object_name*] [, *object-type*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

owner_name Owner of the object. If specified, sp_iqobjectinfo displays output only for tables and join indexes with the specified owner. If not specified, sp_iqobjectinfo displays information on tables and join indexes for all users in the database.

object_name Name of the table or join index. If not specified, sp_iqobjectinfo displays information on all tables and join indexes in the database.

object-type Valid object types are table (the default) or joinindex.

If the object-type is a table, it must be enclosed in quotation marks.

All parameters are optional, and any parameter may be supplied independent of the value of another parameter.

Sybase recommends that you use input parameters with sp_iqobjectinfo; you can query the results of the sp_iqobjectinfo and it performs better if you use input parameters rather than using predicates in the WHERE clause of the query. For example, Query A is written as:

```
SELECT COUNT(*) FROM sp_iqobjectinfo()
WHERE owner = 'DBA'
AND object_name = 'tab_case510'
AND object_type = 'table'
AND sub_object_name is NULL
AND dbspace_name = 'iqmain7'
AND partition_name = 'P1'
```

Query B is Query A rewritten to use sp_iqobjectinfo input parameters:

```
SELECT COUNT(*) FROM
sp_iqobjectinfo('DBA','tab_case510','table')
WHERE sub_object_name is NULL
AND dbspace_name = 'iqmain7'
AND PARTITION_NAME = 'P1'
```

Query B returns results faster than Query A. When the input parameters are passed to sp_iqobjectinfo, the procedure compares and joins fewer records in the system tables, thus doing less work compared to Query A. In Query B, the predicates are applied in the procedure itself and the procedure returns a smaller result set, so a smaller number of predicates is applied in the query.

The sp_iqobjectinfo stored procedure supports wildcard characters for interpreting *owner_name*, *object_name*, and *object_type*. It shows information for all dbspaces that match the given pattern in the same way the LIKE clause matches patterns inside queries.

Description

Returns all the partitions and the dbspace assignments of a particular or all database objects (of type table and join index only) and its sub-objects. The sub-objects are columns, indexes, primary key, unique constraints, and foreign keys.

Table 7-45: sp_iqobjectinfo columns

Column name	Description
owner	Name of the owner of the object.
object_name	Name of the object (of type table and join index only) located on the dbspace.
sub_object_name	Name of the object located on the dbspace.
object_type	Type of the object (column, index, primary key, unique constraint, foreign key, partition, join index or table).
object_id	Global object id of the object.
id	Table id or join-index id of the object.
dbspace_name	Name of the dbspace on which the object resides. The string "[multiple]" is displayed for a special meta row for partitioned objects. The [multiple] row indicates that multiple rows follow in the output to describe the table or column.
partition_name	Name of the partition for the given object.

Examples

Displays information about partitions and dbspace assignments of a specific database object and sub-objects owned by a specific user:

```
sp_iqobjectinfo GROUPO,Departments
```

```
owner   object_name  sub_object_name                object_type  object_id  id
GROUPO  Departments  (NULL)                          table          3632     738
GROUPO  Departments  DepartmentID                    column         3633     738
GROUPO  Departments  DepartmentName                  column         3634     738
GROUPO  Departments  DepartmentHeadID                column         3635     738
GROUPO  Departments  DepartmentsKey                  primary key    83       738
GROUPO  Departments  FK_DepartmentHeadID_EmployeeID  foreign key    92       738

dbspace_name     partition_name
iq_main          (NULL)
iq_main          (NULL)
iq_main          (NULL)
iq_main          (NULL)
iq_main          (NULL)
iq_main          (NULL)
```

Displays information about partitions and dbspace assignments of a specific database object and sub-objects owned by a specific user for *object-type* table:

```
sp_iqobjectinfo DBA,sale,'table'
```

```
owner  object_name  sub_object_name  object_type  object_id  id
DBA    sale          (NULL)           table          3698     742
DBA    sale         prod_id           column         3699     742
DBA    sale         month_num         column         3700     742
DBA    sale         rep_id            column         3701     742
DBA    sale         sales             column         3702     742

dbspace_name      partition_name
iq_main           (NULL)
iq_main           (NULL)
iq_main           (NULL)
iq_main           (NULL)
iq_main           (NULL)
```

sp_iqpassword procedure

Function

Changes a user's password. The preferred way to create a user is by using the CREATE USER statement. See "CREATE USER statement," in Chapter 1, "SQL Statements," in *Reference: Statements and Options*.

Syntax1

call sp_iqpassword ('*caller_password*', '*new_password*' [, '*user_name*'])

Syntax2

sp_iqpassword '*caller_password*', '*new_password*' [, '*user_ name*']

Permissions

None to set your own password; DBA authority required to set other users' passwords.

Usage

caller_password Your password. When you are changing your own password, this is your old password. When the DBA is changing another user's password, caller_password is the DBA's password.

new_password New password for the user, or for *loginname*.

user_name Login name of the user whose password is being changed by the DBA. Do not specify user_name when changing your own password.

Description

Any user can change his or her own password using sp_iqpassword. The DBA can change the password of any existing user.

Examples

Example 1 Changes the password of the logged in user from irk103 to exP984:

```
sp_iqpassword 'irk103', 'exP984'
```

Example 2 Changes the password of user joe from eprr45 to pdi032 only if the logged in user has DBA privileges or the user is joe himself:

```
call sp_iqpassword ('eprr45', 'pdi932', 'joe')
```

sp_iqpkeys procedure

Function

Displays information about primary keys and primary key constraints by table, column, table owner, or for all Sybase IQ tables in the database.

Syntax

sp_iqpkeys { [*table-name*], [*column-name*], [*table-owner*] }

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

table-name The name of a base or global temporary table. If specified, the procedure returns information about primary keys defined on the specified table only.

column-name The name of a column. If specified, the procedure returns information about primary keys on the specified column only.

table-owner The owner of a table or table. If specified, the procedure returns information about primary keys on tables owned by the specified owner only.

One or more of the parameters can be specified. If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. If none of the parameters are specified, a description of all primary keys on all tables in the database is displayed. If any of the specified parameters is invalid, no rows are displayed in the output.

Table 7-46: sp_iqpkeys usage examples

Syntax	Output
sp_iqpkeys sales	Displays information about primary keys defined on table sales
sp_iqpkeys sales, NULL, DBA	Displays information about primary keys defined on table sales owned by DBA
sp_iqpkeys sales, store_id, DBA	Displays information about primary key defined on column store_id of table sales owned by DBA
sp_iqpkeys NULL, NULL, DBA	Displays information about primary keys defined on all tables owned by DBA

Description

The sp_iqpkeys stored procedure displays the following information about primary keys on base and global temporary tables in a database:

Table 7-47: sp_iqpkeys columns

Column name	Description
table_name	The name of the table
table_owner	The owner of the table
column_name	The name of the column(s) on which the primary key is defined
column_id	The column ID
constraint_name	The name of the primary key constraint
constraint_id	The primary key constraint ID

Note The sp_iqpkeys stored procedure exists only in databases created with Sybase IQ version 12.6 or later.

Examples

Display the primary keys defined on columns of table sales1:

```
sp_iqpkeys sales1

table_name table_owner column_name column_id constraint_name constraint_id
sales1         DBA         store_id     1          MA114            114
```

Display the primary keys defined on columns of table sales2:

```
sp_iqpkeys sales2

table_name table_owner column_name column_id constraint_name constraint_id
sales2         DBA       store_id,     1,2        MA115            115
                         order_num
```

Display the primary keys defined on the column store_id of table sales2:

```
sp_iqpkeys sales2, store_id

table_name table_owner column_name column_id constraint_name constraint_id
sales2         DBA       store_id       1          MA115            115
```

See also

"sp_iqindex and sp_iqindex_alt procedures" on page 415

"sp_iqcolumn procedure" on page 370

sp_iqprocedure procedure

Function

Displays information about system and user-defined procedures.

Syntax

sp_iqprocedure [*proc-name*], [*proc-owner*], [*proc-type*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

proc-name The name of the procedure.

proc-owner The owner of the procedure.

proc-type The type of procedure. Allowed values are:

- SYSTEM: displays information about system procedures (procedures owned by user SYS or dbo) only
- ALL: displays information about user and system procedures
- Any other value: displays information about user procedures

The sp_iqprocedure procedure can be invoked without any parameters. If no parameters are specified, only information about user-defined procedures (procedures not owned by dbo or SYS) is displayed by default.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqprocedure NULL, NULL, SYSTEM` and `sp_iqprocedure NULL, user1`.

Table 7-48: sp_iqprocedure usage examples

Syntax	Output
sp_iqprocedure	Displays information about all procedures in the database not owned by dbo or SYS
sp_iqprocedure sp_test	Displays information about the procedure sp_test
sp_iqprocedure non_existing_proc	No rows returned, as the procedure non_existing_proc does not exist
sp_iqprocedure NULL, DBA	Displays information about all procedures owned by DBA
sp_iqprocedure sp_test, DBA	Displays information about the procedure sp_test owned by DBA
sp_iqprocedure sp_iqtable	The procedure sp_iqtable is not a system procedure. If there is no user-defined procedure also named sp_iqtable, no rows are returned. (By default only user-defined procedures are returned.)
sp_iqprocedure sp_iqtable, dbo	No rows returned, as the procedure sp_iqtable is not a user procedure (by default only user procedures returned)
sp_iqprocedure NULL, NULL, SYSTEM	Displays information about all system procedures (owned by dbo or SYS)
sp_iqprocedure sp_iqtable, NULL, ‘YSTEM	Displays information about the system procedure sp_iqtable
sp_iqprocedure sp_iqtable, dbo, ALL	Displays information about the system procedure sp_iqtable owned by dbo

Description

The sp_iqprocedure stored procedure displays information about procedures in a database. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if *proc-name* is specified, only information about the specified procedure is displayed. If *proc-owner* is specified, sp_iqprocedure returns only information about procedures owned by the specified owner. If no parameters are specified, sp_iqprocedure displays information about all the user-defined procedures in the database.

The sp_iqprocedure procedure returns information in the following columns:

Table 7-49: sp_iqprocedure columns

Column name	Description
proc_name	The name of the procedure
proc_owner	The owner of the procedure
proc_defn	The command used to create the procedure. For hidden procedures, the keyword ‘HIDDEN’ is displayed.
replicate	Displays Y if the procedure is a primary data source in a Replication Server installation; N if not.
srvid	Indicates the remote server, if the procedure is on a remote database server
remarks	A comment string

Examples

Displays information about the user-defined procedure sp_test:

```
sp_iqprocedure sp_test

proc_name    proc_owner    proc_defn       replicate     srvid     remarks

sp_test      DBA        create procedure   N             (NULL)    (NULL)
                        DBA.sp_test(in n1
                        integer)
                        begin message‘sp_test’end
```

Displays information about all procedures owned by user DBA:

```
sp_iqprocedure NULL, DBA

proc_name    proc_owner    proc_defn       replicate     srvid     remarks

sp_test      DBA        create procedure   N             (NULL)    (NULL)
                        DBA.sp_test(in n1
                        integer)
                        begin message‘sp_test’end
sp_dept      DBA        create procedure   N             (NULL)    (NULL)
                        DBA.sp_dept() begin end
```

See also

“CREATE USER statement,” in Chapter 1, “SQL Statements,” in *Reference: Statements and Options*

sp_iqprocparm procedure

Function

Displays information about stored procedure parameters, including result set variables and SQLSTATE/SQLCODE error values.

Syntax

sp_iqprocparm [*proc-name*], [*proc-owner*], [*proc-type*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

proc-name The name of the procedure.

proc-owner The owner of the procedure.

proc-type The type of procedure. Allowed values are:

- SYSTEM: displays information about system procedures (procedures owned by user SYS or dbo) only
- ALL: displays information about user and system procedures
- Any other value: displays information about user procedures

You can invoke sp_iqprocparm without parameters. If you do not specify any parameters, input/output and result parameters of user-defined procedures (procedures not owned by dbo or SYS) appear.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqprocparm NULL, NULL, SYSTEM` and `sp_iqprocparm NULL, user1`.

Table 7-50: sp_iqprocparm usage examples

Syntax	Output
sp_iqprocparm	Displays parameters for all procedures in the database not owned by dbo or SYS
sp_iqprocparm sp_test	Displays information about the procedure sp_test
sp_iqprocparm non_existing_proc	No rows returned, as the procedure non_existing_proc does not exist
sp_iqprocparm NULL, DBA	Displays parameters for all procedures owned by DBA
sp_iqprocparm sp_test, DBA	Displays parameters for the procedure sp_test owned by DBA
sp_iqprocparm sp_iqtable	sp_iqtable is a system procedure. If there is no user-defined procedure also named sp_iqtable, no rows are returned. (By default, only user-defined procedures are returned.)
sp_iqprocparm sp_iqtable, dbo	No rows returned, as the procedure sp_iqtable is not a user procedure. (By default, only user procedures are returned.)
sp_iqprocparm NULL, NULL, SYSTEM	Displays parameters for all system procedures (owned by dbo or SYS)
sp_iqprocparm sp_iqtable, NULL, SYSTEM	Displays parameters of the system procedure sp_iqtable
sp_iqprocparm sp_iqtable, dbo, ALL	Displays parameters of the system procedure sp_iqtable owned by dbo

Description

The sp_iqprocparm stored procedure displays information about stored procedure parameters, including result set variables and SQLSTATE/SQLCODE error values. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if *proc-name* is specified, only information about parameters to the specified procedure displays. If *proc-owner* is specified, sp_iqprocparm only returns information about parameters to procedures owned by the specified owner. If no parameters are specified, sp_iqprocparm displays information about parameters to all the user-defined procedures in the database.

The sp_iqprocparm procedure returns information in the following columns:

Table 7-51: sp_iqprocparm columns

Column name	Description
proc_name	The name of the procedure
proc_owner	The owner of the procedure
parm_name	The name of the parameter
parm_type	The type of parameter is one of the following values: • normal parameter (variable) • result variable: used with procedures that return result sets • SQLSTATE error value • SQLCODE error value
parm_mode	The mode of the parameter: whether a parameter supplies a value to the procedure, returns a value, does both, or does neither. Parameter mode is one of the following: • in: parameter supplies a value to the procedure • out: parameter returns a value • inout: parameter supplies as well as returns a value • NULL: parameter neither supplies nor returns a value
domain_name	The name of the data type of the parameter as listed in the SYSDOMAIN system table
width	The length of string parameters, the precision of numeric parameters, and the number of bytes of storage for all other data types
scale	The number of digits after the decimal point for numeric data type parameters and zero for all other data types
default	The default value of the parameter, held as a string

Examples

Display information about the parameters of the user-defined procedure sp_test:

```
sp_iqprocparm sp_test

proc_name proc_owner parm_name parm_type parm_mode domain_name width scale  default

sp_test   DBA        ID         normal    in        integer     4     0      (NULL)
```

Display information about the parameters of the system procedure sp_iqshowcompression:

```
sp_iqprocparm sp_iqshowcompression, dbo, system

proc_name            proc_owner  parm_name    parm_type  parm_mode
domain_name  width  scale  default
```

```
sp_iqshowcompression  dbo        @owner_name  normal     in
char         128    0      (NULL)
sp_iqshowcompression  dbo        @table_name  normal     in
char         128    0      (NULL)
sp_iqshowcompression  dbo        @column_name normal     in
char         128    0      (NULL)
sp_iqshowcompression  dbo        Column       result     out
char         128    0      (NULL)
sp_iqshowcompression  dbo        Compression  result     out
char         3      0      (NULL)
```

See also

CREATE PROCEDURE statement in Chapter 1, “SQL Statements,” in *Reference: Statements and Options*

sp_iqrebuildindex procedure

Function

Rebuilds one or more indexes on a table with the original IQ UNIQUE value specified in the CREATE TABLE statement, or a new IQ UNIQUE value to change storage required and/or query performance. To rebuild an index other than the default index, specify the index name.

Syntax

sp_iqrebuildindex (*table_name*, *index_clause*)

Permissions

You must have INSERT permission on a table to rebuild an index on that table.

Usage

table_name Partial or fully qualified table name on which the index rebuild process takes place. If the user both owns the table and executes the procedure, a partially qualified name may be used; otherwise, the table name must be fully qualified.

index_clause One or more of the following strings, separated by spaces:

column *column_name [count]*

[dbspace *dbspace_name*]

index *index_name*

[dbspace *dbspace_name*]

Each *column_name* or *index_name* must refer to a column or index on the specified table. If you specify a *column_name* or *index_name* multiple times, the procedure returns an error and no index is rebuilt.

The *count* is a nonnegative number that represents the IQ UNIQUE value. In a CREATE TABLE statement, IQ UNIQUE (count) approximates how many distinct values can be in a given column. The number of distinct values affects query speed and storage requirements. For details, see “Optimizing storage and query performance,” in Chapter 5, “Working with Database Objects” in the *System Administration Guide: Volume 1*.

You must specify the keywords column and index. The keyword dbspace is optional. These keywords are not case-sensitive.

Sybase IQ rebuilds the column or index in the same dbspace where the original resided unless you specify dbspace *dbspace-name*.

See also

“sp_iqindexfragmentation procedure” on page 419, and “sp_iqrowdensity procedure” on page 456.

Chapter 6, “Using Sybase IQ Indexes” in *System Administration Guide: Volume 1*.

“FP_LOOKUP_SIZE option” and “MINIMIZE_STORAGE option” in Chapter 2, “Database Options” in *Reference: Statements and Options.*

Description

If you specify a column name, the procedure rebuilds the default index for that column, and no index name is needed. Specifying the name of the default index assigned by Sybase IQ in addition to the column name in this situation returns an error. If you omit *count* after the *column_name*, value 0 (zero) is used as the default.

If the default index is a one-byte index, sp_iqrebuildindex always rebuilds it as a one-byte index no matter what IQ UNIQUE value the user specified.

For one-byte default indexes, if the specified value in *column_name (count)* is 0 or greater than 256, the column’s cardinality value is used to update the approx_unique_count column in SYS.SYSIQCOLUMN.

If the column has the data type VARCHAR or VARBINARY greater than 255 bytes, sp_iqrebuildindex will not rebuild a default index.

sp_iqrebuildindex rebuilds a WD index on a column of data type LONG VARCHAR (CLOB).

If the default index is a two-byte index, and the specified count is 0 or greater than 65536, the column’s cardinality value determines whether to rebuild the default into a one-byte or two-byte index, and that value is used to update the approx_unique_count column in SYS.SYSIQCOLUMN.

If you specify a nonzero IQ UNIQUE value, the default index is rebuilt as a one-byte, two-byte, or flat default index, with exceptions described above.

If you specify an IQ UNIQUE value of zero or no IQ UNIQUE value, the MINIMIZE_STORAGE option controls how the index is rebuilt:

- If MINIMIZE_STORAGE option is set ON, the index is rebuilt as a one-byte default index first, and converted to two-byte or flat if necessary.
- If MINIMIZE_STORAGE is set OFF, the index is rebuilt using the default for the data type. For more information, see “Sybase IQ index types” Chapter 7, “System Procedures” in the *System Administration Guide: Volume 1*.

Examples

Rebuilds the default index on column *Surname*:

```
sp_iqrebuildindex ‘emp1‘, ‘column dept_id‘
```

or:

```
call sp_iqrebuildindex (‘empl1‘, ‘column dept_id‘)
```

Creates a flat default index on column *c1*:

```
CREATE TABLE mytable (c1 int IQ UNIQUE 1000000000)
```

Converts the default one-byte index to a two-byte index:

```
sp_iqrebuildindex ‘mytable‘, ‘column c1 1024‘
```

or:

```
call sp_iqrebuildindex (‘mytable‘, ‘column c1 1024‘)
```

See also

“sp_iqindexfragmentation procedure” on page 419, and “sp_iqrowdensity procedure” on page 456.

“FP_LOOKUP_SIZE option” and “MINIMIZE_STORAGE option” in Chapter 2, “Database Options” in *Reference: Statements and Options*.

Chapter 6, “Using Sybase IQ Indexes” in *System Administration Guide: Volume 1*.

sp_iqrename procedure

Function

Renames user-created tables, columns, indexes, constraints (unique, primary key, foreign key, and check), stored procedures, and functions.

Syntax

sp_iqrename *object-name*, *new-name* [, *object-type*]

Permissions

Must be the owner of the table or have DBA authority or alter permission on the object. Requires exclusive access to the object.

Usage

object-name The original name of the user-created object.

Optionally, *owner-name* can be specified as part of *object-name* as *owner-name.object-name,* where *owner-name* is the name of the owner of the object being renamed. If *owner-name* is not specified, the user calling sp_iqrename is assumed to be the owner of the object. Note that the object is successfully renamed only if the user calling sp_iqrename has the required permissions to rename the object.

If the object to be renamed is a column, index, or constraint, you *must* specify the name of the table with which the object is associated. For a column, index, or constraint, *object-name* can be of the form *table-name.object-name* or *owner-name.table-name.object-name*.

new-name The new name of the object. The name must conform to the rules for identifiers and must be unique for the type of object being renamed.

object-type An optional parameter that specifies the type of the user-created object being renamed, that is, the type of the object *object-name*. The *object-type* parameter can be specified in either upper or lowercase.

Table 7-52: sp_iqrename object-type parameter values

object-type parameter	Specifies
column	The object being renamed is a column
index	The object being renamed is an index
constraint	The object being renamed is a unique, primary key, check, or referential (foreign key) constraint
procedure	The object being renamed is a function
object-type not specified	The object being renamed is a table

---

Warning! You must change appropriately the definition of any dependent object (procedures, functions, and views) on an object being renamed by sp_iqrename. The sp_iqrename procedure does not automatically update the definitions of dependent objects. You must change these definitions manually.

Description

The sp_iqrename stored procedure renames user-created tables, columns, indexes, constraints (unique, primary key, foreign key, and check), and functions.

If you attempt to rename an object with a name that is not unique for that type of object, sp_iqrename returns the message “Item already exists.”

sp_iqrename does not support renaming a view, a procedure, an event or a data type. The message “Feature not supported.” is returned by sp_iqrename, if you specify event or datatype as the *object-type* parameter.

Examples

Renames the table titles owned by user shweta to books:

```
sp_iqrename shweta.titles, books
```

Renames the column id of the table books to isbn:

```
sp_iqrename shweta.books.id, isbn, column
```

Renames the index idindex on the table books to isbnindex:

```
sp_iqrename books.idindex, isbnindex, index
```

Renames the primary key constraint prim_id on the table books to prim_isbn:

```
sp_iqrename books.prim_id, prim_isbn, constraint
```

See also

ALTER TABLE statement RENAME clause and ALTER INDEX statement RENAME clause in *Reference: Statements and Options*

sp_iq_reset_identity procedure

Function

Sets the seed of the Identity/Autoincrement column associated with the specified table to the specified value.

Syntax

sp_iq_reset_identity (*table_name*, *table_owner*, *value*)

Usage

Syntax You must specify *table_name*, *table owner*, and *value*.

Permissions

None required.

Description

The Identity/Autoincrement column stores a number that is automatically generated. The values generated are unique identifiers for incoming data. The values are sequential, are generated automatically, and are never reused, even when rows are deleted from the table. The seed value specified replaces the default seed value and persists across database shutdowns and failures.

Example

The following example creates an Identity column with a starting seed of 50.:

```
CREATE TABLE mytable(c1 INT identity)
call sp_iq_reset_identity('mytable', 'dba', 50)
```

See also

"sp_iqcolumn procedure" on page 370

sp_iqrestoreaction procedure

Function

Shows what restore actions are needed to bring database to a consistent state with a given past date.

Syntax

sp_iqrestoreaction [*timestamp*]

Parameters

timestamp Specifies the past date target.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

sp_iqrestoreaction returns an error if the database cannot be brought to a consistent state for the timestamp. Otherwise, suggests restore actions that will return the database to a consistent state.

The common point to which the database can be restored coincides with the last backup time that backed up read-write files just before the specified timestamp. The backup may be all-inclusive or read-write files only.

Output may not be in exact ascending order based on backup time. If a backup archive consists of multiple read-only dbfiles, it may contain multiple rows (with the same backup time and backup id).

If you back up a read-only dbfile or dbspace multiple times, the restore uses the last backup. The corresponding backup time could be after the specified timestamp, as long as the dbspace/dbfile alter ID matches the dbspace/dbfile alter ID recorded in the last read-write backup that is restored.

sp_iqrestoreaction returns the following:

Table 7-53: sp_iqrestoreaction columns

Column name	Description
sequence_number	Orders the steps to be taken
backup_id	Identifier for the backup transaction
backup_archive_list	List of archive files in the backup
backup_time	Time of the backup taken
virtual_type	Type of virtual backup: “Non-virtual,” “Decoupled,” or “Encapsulated”
restore_dbspace	Can be empty. Indicates that all dbspaces are to be restored from the backup archive
restore_dbfile	Could be empty. Indicates that all dbfiles in the given dbspace are to be restored from the backup archive
backup_comment	User comment

Example

Sample output of sp_iqrestoreaction:

```
sequence_number   backup_id   backup_archive_list    backup_time
             1        1192   c:\\\\temp\\\\b1        2008-09-23 14:47:40.0
             2        1201   c:\\\\temp\\\\b2.inc    2008-09-23 14:48:05.0l
             3        1208   c:\\\\temp\\\\b3.inc    2008-09-23 14:48:13.0

virtual_type   restore_dbspace   restore_dbfile   backup_comment
Nonvirtual
```

```
Nonvirtual
Nonvirtual
```

sp_iqrowdensity procedure

Function

Reports information about the internal row fragmentation for a table at the FP index level.

Syntax

dbo.sp_iqrowdensity (‘*target* ‘)

target:(**table** *table-name* | (**column** *column-name* (...))

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

table-name Target table *table-name* reports on all columns in the named table.

column-name Target column *column-name* reports on the named column in the target table. You may specify multiple target columns, but must repeat the keyword each time.

You must specify the keywords table and column. These keywords are not case-sensitive.

Description

sp_iqrowdensity measures row fragmentation at the default index level. Density is the ratio of the minimum number of pages required by an index for existing table rows to the number of pages actually used by the index. This procedure returns density as a number such that $0 < density \leq 1$. For example, if an index that requires 8 pages minimum storage occupies 10 pages, its density is .8.

The density reported does not indicate the number of disk pages that may be reclaimed by re-creating or reorganizing the default index.

This procedure displays information about the row density of a column, but does not recommend further action. You must determine whether or not to re-create, reorganize, or rebuild an index.

Example

Reports the row density on column *ID* in table *SalesOrders*:

```
sp_iqrowdensity('column groupo.SalesOrders.ID')
```

Tablename	ColumnName	IndexType	Density
'GROUPO.SalesOrders'	'ID'	'Flat style FP'	'1.0'

See also

“FP_LOOKUP_SIZE option” and “MINIMIZE_STORAGE option” in Chapter 2, “Database Options” in *Reference: Statements and Options*.

Chapter 6, "Using Sybase IQ Indexes" in *System Administration Guide: Volume 1*.

sp_iqshowpsexe procedure

Function

Displays information about the settings of database options that control the priority of tasks and resource usage for connections.

Syntax

sp_iqshowpsexe [*connection-id*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

connection-id An integer representing the connection ID.

If *connection-id* is specified, sp_iqshowpsexe returns information only about the specified connection. If *connection-id* is not specified, sp_iqshowpsexe returns information about all connections.

If the specified *connection-id* does not exist, sp_iqshowpsexe returns no rows.

Description

The sp_iqshowpsexe stored procedure displays information about the settings of database options that control the priority of tasks and resource usage for connections, which is useful to database administrators for performance tuning.

Table 7-54: sp_iqshowpsexe columns

Column name	Description
connectionid	The connection ID
application	Information about the client application that opened the connection. Includes the AppInfo connection property information: HOST: the host name of the client machine EXE: the name of the client executable (Windows only) APPINFO: the APPINFO in the client connection string, if specified
userid	Login name of the user that opened the connection
iqgovern_priority	Value of the database option IQGOVERN_PRIORITY that assigns a priority to each query waiting in the -iqgovern queue. By default, this option has a value of 2 (MEDIUM). The values 1, 2, and 3 are shown as HIGH, MEDIUM, and LOW, respectively.
max_query_time	Value of the database option MAX_QUERY_TIME that sets a limit, so that the optimizer can disallow very long queries. By default, this option is disabled and has a value of 0.
query_row_limit	Value if the database option QUERY_ROWS_RETURNED_LIMIT that sets the row threshold for rejecting queries based on the estimated size of the result set. The default is 0, which means there is no limit.
query_temp_space_limit	Value of the database option QUERY_TEMP_SPACE_LIMIT (in MB) that constrains the use of temporary IQ dbspace by user queries. The default value is 2000MB.
max_cursors	Value of the database option MAX_CURSOR_COUNT that specifies a resource governor to limit the maximum number of cursors a connection can use at once. The default value is 50. A value of 0 implies no limit.
max_statements	Value of the database option MAX_STATEMENT_COUNT that specifies a resource governor to limit the maximum number of prepared statements that a connection can use at once. The default value is 100. A value of 0 implies no limit.

Note The AppInfo property may not be available from Open Client or jConnect applications such as the Java version of Interactive SQL (dbisql) or Sybase Central. If the AppInfo property is not available, the application column is blank.

Example

Display information about the settings of database options that control the priority of tasks and resource usage for connection ID 2:

```
sp_iqshowpsexe 2

connectionid    application
           2    HOST=GOODGUY-XP;EXE=C:\\Program Files\\Sybase\\
                IQ-15_1\\bin32\\dbisqlg.exe;

userid    iqgovern_priority    max_query_time    query_row_limit
DBA       MEDIUM                            0                  0

query_temp_space_limit    max_statements    max_cursors
                  2000                50            100
```

See also

In Chapter 7, "System Procedures": "sp_iqconnection procedure" on page 373, "sp_iqcontext procedure" on page 378, and "sa_conn_info system procedure" on page 494

"CONNECTION_PROPERTY function [System]" on page 139

In Chapter 2, "Database Options" in *Reference: Statements and Options*: IQGOVERN_MAX_PRIORITY option, IQGOVERN_PRIORITY option, IQGOVERN_PRIORITY_TIME option, MAX_QUERY_TIME option, QUERY_ROWS_RETURNED_LIMIT option, QUERY_TEMP_SPACE_LIMIT option, MAX_CURSOR_COUNT option, and MAX_STATEMENT_COUNT option

"AppInfo connection parameter [App]" in Chapter 4, "Connection and Communication Parameters" in the *System Administration Guide: Volume 1*

sp_iqspaceinfo procedure

Function

Displays the number of blocks used by each object in the current database and the name of the dbspace in which the object is located.

Syntax

sp_iqspaceinfo ['**main**
| [**table** *table-name* **|** **index** *index-name*] [...] ']

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

For the current database, displays the object name, number of blocks used by each object, and the name of the dbspace. sp_iqspaceinfo requires no parameters.

The information returned by sp_iqspaceinfo is helpful in managing dbspaces.

Example — The following output is from the sp_iqspaceinfo stored procedure run on the iqdemo database. Output for some tables and indexes have been removed in this example.

```
             Name                                  NBlocks    dbspace_name
Contacts                                             19       IQ_SYSTEM_MAIN
SalesOrderItems.DBA.ASIQ_IDX_T205_C5_FP              56       IQ_SYSTEM_MAIN
Contacts.DBA.ASIQ_IDX_T206_C10_FP                    55       IQ_SYSTEM_MAIN
Contacts.DBA.ASIQ_IDX_T206_C1_FP                     61       IQ_SYSTEM_MAIN
...
Contacts.DBA.ASIQ_IDX_T206_C9_FP                     55       IQ_SYSTEM_MAIN
Contacts.DBA.ASIQ_IDX_T206_I11_HG                    19       IQ_SYSTEM_MAIN
Customers                                            20       IQ_SYSTEM_MAIN
Customers.DBA.ASIQ_IDX_T207_C1_FP                    61       IQ_SYSTEM_MAIN
Customers.DBA.ASIQ_IDX_T207_C2_FP                    55       IQ_SYSTEM_MAIN
...
Customers.DBA.ASIQ_IDX_T207_I10_HG                   19       IQ_SYSTEM_MAIN
...
```

See also — “sp_iqindexinfo procedure” on page 421, “sp_iqdbspace procedure” on page 388, and “sp_iqdbspaceinfo procedure” on page 391

Chapter 5, “Working with Database Objects” in the *System Administration Guide: Volume 1*

sp_iqspaceused procedure

Function — Shows information about space available and space used in the IQ store and IQ temporary store.

Syntax

sp_iqspaceused(out mainKB unsigned bigint,
out mainKBUsed unsigned bigint,
out tempKB unsigned bigint,
out tempKBUsed unsigned bigint)

Permissions — DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage — sp_iqspaceused returns four values as unsigned bigint out parameters. This system stored procedure can be called by user-defined stored procedures to determine the amount of main and temporary IQ store space in use.

Description — sp_iqspaceused returns a subset of the information provided by sp_iqstatus, but allows the user to return the information in SQL variables to be used in calculations.

Table 7-55: sp_iqspaceused columns

Column name	Description
mainKB	The total IQ main store space in kilobytes.
mainKBUsed	The number of kilobytes of IQ main store space used by the database.
tempKB	The total IQ temporary store space in kilobytes.
tempKBUsed	The number of kilobytes of IQ temporary store space in use by the database.

Example

sp_iqspaceused requires four output parameters. The following example shows the creation of a user-defined stored procedure myspace that declares the four output parameters and then calls sp_iqspaceused:

```
create procedure dbo.myspace()
begin
  declare mt unsigned bigint;
  declare mu unsigned bigint;
  declare tt unsigned bigint;
  declare tu unsigned bigint;
  call sp_iqspaceused(mt,mu,tt,tu);
  select cast(mt/1024 as unsigned bigint) as mainMB,
        cast(mu/1024 as unsigned bigint) as mainusedMB,
        mu*100/mt as mainPerCent,
        cast(tt/1024 as unsigned bigint) as tempMB,
        cast(tu/1024 as unsigned bigint) as tempusedMB,
        tu*100/tt as tempPerCent;
end
```

To display the output of sp_iqspaceused, run the procedure myspace:

```
myspace
```

sp_iqstatistics procedure

Function

Returns serial number, name, description, value, and unit specifier for each available statistic, or a specified statistic.

Syntax

sp_iqstatistics [*stat_name*]

Parameters

stat_name (Optional) VARCHAR parameter specifying the name of a statistic.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

When stat_name is provided, sp_iqstatistics returns one row for the given statistic, or zero rows if the name is invalid. When invoked without any parameter, sp_iqstatistics returns all statistics.

Result set

Column name	Data type	Description
stat_num	UNSIGNED INTEGER	Serial number of a statistic
stat_name	VARCHAR(255)	Name of statistic
stat_desc	VARCHAR(255)	Description of statistic
stat_value	LONG VARCHAR	Value of statistic
stat_unit	VARCHAR(128)	Unit specifier

The following statistics may be returned:

stat_ num	stat_name	stat_desc	stat_unit
0	CpuTotalTime	Total CPU time in seconds consumed by the IQ server since last server startup	Second
1	CpuUserTime	CPU user time in seconds consumed by the IQ server since last server startup	Second
2	CpuSystemTime	CPU system time in seconds consumed by the IQ server since last server startup	Second
3	ThreadsFree	Number of IQ threads free	N/A
4	ThreadsInUse	Number of IQ threads in use	N/A
5	MemoryAllocated	Allocated memory in megabytes	MB
6	MemoryMaxAllocated	Max allocated memory in megabytes	MB
7	MainCacheCurrentSize	Main cache current size in megabytes	MB
8	MainCacheFinds	Main cache total number of lookup requests	N/A
9	MainCacheHits	Main cache total number of hits	N/A
10	MainCachePagesPinned	Main cache number of pages pinned	Page
11	MainCachePagesPinnedPercentage	Percentage of main cache pages pinned	%
12	MainCachePagesDirtyPercentage	Percentage of main cache pages dirtied	%
13	MainCachePagesInUsePercentage	Percentage of main cache pages in use	%
14	TempCacheCurrentSize	Temporary cache current size in megabytes	MB
15	TempCacheFinds	Temporary cache total number of lookup requests	N/A
16	TempCacheHits	Temporary cache total number of hits	N/A
17	TempCachePagesPinned	Temporary cache number of pages pinned	Page

stat_ num	stat_name	stat_desc	stat_unit
18	TempCachePagesPinnedPercentage	Percentage of temporary cache pages pinned	%
19	TempCachePagesDirtyPercentage	Percentage of temporary cache pages dirtied	%
20	TempCachePagesInUsePercentage	Percentage of temporary cache pages in use	%
21	MainStoreDiskReads	Number of kilobytes read from main store	KB
22	MainStoreDiskWrites	Number of kilobytes written to main store	KB
23	TempStoreDiskReads	Number of kilobytes read from main store	KB
24	TempStoreDiskWrites	Number of kilobytes written to main store	KB
25	ConnectionsTotalConnections	Total number of connections since server startup	N/A
26	ConnectionsTotalDisonnections	Total number of disconnections since server startup	N/A
27	ConnectionsActive	Number of active connections	N/A
28	OperationsWaiting	Number of operations waiting for IQ resource governor	N/A
29	OperationsActive	Number of active concurrent operations admitted by IQ resource governor	N/A
30	OperationsActiveLoadTableStatements	Number of active LOAD TABLE statements	N/A

Examples

Example 1 Displays a single statistic, the total CPU time:

```
sp_iqstatistics 'CPUTotalTime'
```

Example 2 Displays all statistics for MainCache%:

```
SELECT * from sp_iqstatistics() WHERE stat_name LIKE
'MainCache%'
```

sp_iqstatus procedure

Function

Displays a variety of Sybase IQ status information about the current database.

Syntax

sp_iqstatus

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Shows status information about the current database, including the database name, creation date, page size, number of dbspace segments, block usage, buffer usage, I/O, backup information, and so on.

sp_iqstatus displays an out-of-space status for main and temporary stores. If a store runs into an out-of-space condition, sp_iqstatus shows `Y` in the store's out-of-space status display value.

sp_iqspaceused returns a subset of the same information as provided by sp_iqstatus, but allows the user to return the information in SQL variables to be used in calculations. See "sp_iqspaceused procedure" on page 460.

To display space that can be reclaimed by dropping connections, use sp_iqstatus and add the results from the two returned rows:

```
(DBA)> select * from sp_iqstatus() where name like
'%Versions:%'
Execution time: 6.25 seconds
Name                     Value
----------------------------
Other Versions: 2 = 1968Mb
Active Txn Versions: 1 = C:2175Mb/D:2850Mb

(First 2 rows)
```

The above example output shows that one active write transaction created 2175MB and destroyed 2850 MB of data. The total data consumed in transactions and not yet released is 4818MB, or 1968MB + 2850MB = 4818MB.

sp_iqstatus does not show blocks that will be deallocated at the next checkpoint. These blocks do however, appear in sp_iqdbspace output as type X.

Example

The following output is from the sp_iqstatus stored procedure:

```
Sybase IQ (TM)                  Copyright (c) 1992-2009 by Sybase, Inc.
                                         All rights reserved.
Version:                                 15.1.0/090416/P/MS/Windows/2003/
                                         32bit/2009-04-16 02:11:41
Time Now:                                2009-04-21 13:48:22.319
Build Time:                              2009-04-16 02:15:39
File Format:                             23 on 03/18/1999
Server mode:                             IQ Server
Catalog Format:                          2
Stored Procedure Revision:               1
Page Size:                               131072/8192blksz/16bpp
Number of Main DB Files :                2
Main Store Out Of Space:                 N
Number of Temp DB Files :                1
Temp Store Out Of Space:                 N
DB Blocks:  1-3200                       IQ_SYSTEM_MAIN
DB Blocks:  1045440-1055039              iq_main
Temp Blocks:  1-1600                     IQ_SYSTEM_TEMP
```

```
Create Time:                              2009-04-03 11:30:20.674
Update Time:                              2009-04-03 11:34:33.040
Main IQ Buffers:                          255, 32Mb
Temporary IQ Buffers:                     191, 24Mb
Main IQ Blocks Used:                      5915 of 11200, 52%=46Mb, Max Block#:105278
Temporary IQ Blocks Used:                 65 of 800, 8%=0Mb, Max Block#: 0
Main Reserved Blocks Available:           1600 of 1600, 100%=6Mb
Temporary Reserved Blocks Available:      6400 of 6400, 100%=50Mb
IQ Dynamic Memory:                        Current: 69mb, Max: 70mb
Main IQ Buffers:                          Used: 17, Locked: 0
Temporary IQ Buffers:                     Used: 4, Locked: 0
Main IQ I/O:                              I: L1581/P14 O: C3/D163/P161 D:34 C:97.1
Temporary IQ I/O:                         I: L6627/P0 O: C1086/D1166/P83
                                          D:1082 C:100.0
Other Versions:                           0 = 0Mb
Active Txn Versions:                      0 = C:0Mb/D:0Mb
Last Full Backup ID:                      0
Last Full Backup Time:
Last Backup ID:                           0
Last Backup Type:                         None
Last Backup Time:
DB Updated:                               1
Blocks in next ISF Backup:                0 Blocks: =0Mb
Blocks in next ISI Backup:                0 Blocks: =0Mb
DB File Encryption Status:                OFF
```

The following is a key to understanding the `Main IQ I/O` and `Temporary IQ I/O` output codes:

- `I:` Input
- `L:` Logical pages read ("Finds")
- `P:` Physical pages read
- `O:` Output
- `C:` Pages created
- `D:` Pages dirtied
- `P:` Physically written
- `D:` Pages destroyed
- `C:` Compression ratio

See also sp_iqtransaction procedure and sp_iqversionuse procedure

sp_iqsysmon procedure

Function

Monitors multiple components of Sybase IQ, including the management of buffer cache, memory, threads, locks, I/O functions, and CPU utilization.

Batch mode syntax

sp_iqsysmon start_monitor
sp_iqsysmon stop_monitor [, "section(s)"]
or
sp_iqsysmon "*time-period*" [, "section(s)"]

File mode syntax

sp_iqsysmon start_monitor, 'filemode' [, "*monitor-options*"]
sp_iqsysmon stop_monitor

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Batch mode usage

start_monitor Starts monitoring.

stop_monitor Stops monitoring and displays the report.

time-period The time period for monitoring. Must be in the form HH:MM:SS.

section(s) The abbreviation for one or more sections to be displayed by sp_iqsysmon. When more than one section is specified, the section abbreviations must be separated by spaces and the list must be enclosed in single or double quotes. The default is to display all sections.

For the sections related to IQ store, you can specify main or temporary store by prefixing the section abbreviation with "m" or "t", respectively. See Table 7-56. Without the prefix, both stores are monitored. For example, if you specify "mbufman", only the IQ main store buffer manager is monitored. If you specify "mbufman tbufman" or "bufman", both the main and temporary store buffer managers are monitored.

Table 7-56: sp_iqsysmon report section abbreviations

Report section or IQ component	Abbreviation
Buffer manager	(m/t)bufman
Buffer pool	(m/t)bufpool
Prefetch management	(m/t)prefetch
Free list management	(m/t)freelist
Buffer allocation	(m/t)bufalloc
Memory management	memory
Thread management	threads
CPU utilization	cpu
Transaction management	txn
Server context statistics	server
Catalog statistics	catalog

Note The Sybase IQ components Disk I/O and lock manager are not currently supported by sp_iqsysmon.

File mode usage

start_monitor Starts monitoring.

stop_monitor Stops monitoring and writes the remaining output to the log file.

filemode Specifies that sp_iqsysmon is running in file mode. In file mode, a sample of statistics is displayed for every interval in the monitoring period. By default, the output is written to a log file named *dbname.connid-iqmon*. Use the file_suffix option to change the suffix of the output file. See the *monitor_options* parameter for a description of the file_suffix option.

monitor_options The *monitor _options* string can include one or more of the following options:

- -interval *seconds*
  Specifies the reporting interval in seconds. A sample of monitor statistics is output to the log file after every interval. The default is every 60 seconds, if the -interval option is not specified. The minimum reporting interval is 2 seconds. If the interval specified for this option is invalid or less than 2 seconds, the interval is set to 2 seconds.

The first display shows the counters from the start of the server. Subsequent displays show the difference from the previous display. You can usually obtain useful results by running the monitor at the default interval of 60 seconds during a query with performance problems or during a time of day with performance problems. A very short interval may not provide meaningful results. The interval should be proportional to the job time; 60 seconds is usually more than enough time.

- -file_suffix *suffix*
  Creates a monitor output file named *dbname.connid-suffix*. If you do not specify the -file_suffix option, the suffix defaults to *iqmon*. If you specify the -file_suffix option and do not provide a suffix or provide a blank string as a suffix, no suffix is used.

- -append or -truncate
  Directs sp_iqsysmon to append to the existing output file or truncate the existing output file, respectively. Truncate is the default. If both options are specified, the option specified later in the string is effective.

- -section *section(s)*
  Specifies the abbreviation of one or more sections to write to the monitor log file. The default is to write all sections. The abbreviations specified in the sections list in file mode are the same abbreviations used in batch mode. See Table 7-56 for a list of abbreviations. When more than one section is specified, spaces must separate the section abbreviations.

  If the -section option is specified with no sections, none of the sections are monitored. An invalid section abbreviation is ignored and a warning is displayed in the IQ message file.

Usage syntax examples

Table 7-57: sp_iqsysmon usage examples

Syntax	Result
sp_iqsysmon start_monitor sp_iqsysmon stop_monitor	Starts the monitor in batch mode and displays all sections for main and temporary store
sp_iqsysmon start_monitor sp_iqsysmon stop_monitor "mbufman mbufpool"	Starts the monitor in batch mode and displays the Buffer Manager and Buffer Pool statistics for main store
sp_iqsysmon "00:00:10", "mbufpool memory"	Runs the monitor in batch mode for 10 seconds and displays the consolidated statistics at the end of the time period
sp_iqsysmon start_monitor, 'filemode', "-interval 5 -sections mbufpool memory" sp_iqsysmon stop_monitor	Starts the monitor in file mode and writes to the log file every 5 seconds the statistics for Main Buffer Pool and Memory Manager

Description

The sp_iqsysmon stored procedure monitors multiple components of Sybase IQ, including the management of buffer cache, memory, threads, locks, I/O functions, and CPU utilization.

The sp_iqsysmon procedure supports two modes of monitoring:

- Batch mode

  In batch mode, sp_iqsysmon collects the monitor statistics for the period between starting and stopping the monitor or for the time period specified in the *time-period* parameter. At the end of the monitoring period, sp_iqsysmon displays a list of consolidated statistics.

  sp_iqsysmon in batch mode is similar to the Adaptive Server Enterprise procedure sp_sysmon.

- File mode

  In file mode, sp_iqsysmon writes the sample statistics in a log file for every interval period between starting and stopping the monitor.

  Note that the first display in file mode shows the counters from the start of the server. Subsequent displays show the difference from the previous display.

  sp_iqsysmon in file mode is similar to the IQ UTILITIES command START MONITOR and STOP MONITOR interface.

Batch mode examples

Prints monitor information after 10 minutes:

```
sp_iqsysmon "00:10:00"
```

Prints only the Memory Manager section of the sp_iqsysmon report after 5 minutes:

```
sp_iqsysmon “00:05:00”, memory
```

Starts the monitor, executes two procedures and a query, stops the monitor, then prints only the Buffer Manager section of the report:

```
sp_iqsysmon start_monitor
go
execute proc1
go
execute proc2
go
select sum(total_sales) from titles
go
sp_iqsysmon stop_monitor, bufman
go
```

Prints only the Main Buffer Manager and Main Buffer Pool sections of the report after 20 minutes:

```
sp_iqsysmon “00:02:00”, “mbufman mbufpool”
```

File mode examples

Truncates and writes information to the log file every 2 seconds between starting the monitor and stopping the monitor:

```
sp_iqsysmon start_monitor, ‘filemode’, ‘-interval 2’
.
.
.
sp_iqsysmon stop_monitor
```

Appends output for only the Main Buffer Manager and Memory Manager sections to an ASCII file with the name *dbname.connid-testmon*. For the database iqdemo, writes results in the file *iqdemo.2-testmon*:

```
sp_iqsysmon start_monitor, ‘filemode’,
“-file_suffix testmon -append -section mbufman memory”
.
.
.
sp_iqsysmon stop_monitor
```

Example

Run the monitor in batch mode for 10 seconds and display the consolidated statistics at the end of the time period

```
sp_iqsysmon “00:00:10”, “mbufpool memory”

==============================
```

```
Buffer Pool (Main)
==============================
STATS-NAME          TOTAL  NONE  BTREEV  BTREEF  BV  VDO  DBEXT  DBID  SORT
MovedToMRU          0      0     0       0       0   0    0      0     0
MovedToWash         0      0     0       0       0   0    0      0     0
RemovedFromLRU      0      0     0       0       0   0    0      0     0
RemovedFromWash     0      0     0       0       0   0    0      0     0
RemovedInScanMode   0      0     0       0       0   0    0      0     0

STORE  GARRAY  BARRAY  BLKMAP  HASH  CKPT  BM  TEST  CMID  RIDCA  LOB
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0

STATS-NAME                     VALUE
Pages                          127   ( 100.0 %)
InUse                          4     ( 3.1 %)
Dirty                          1     ( 0.8 %)
Pinned                         0     ( 0.0 %)
Flushes                        0
FlushedBufferCount             0
GetPageFrame                   0
GetPageFrameFailure            0
GotEmptyFrame                  0
Washed                         0
TimesSweepersWoken             0

washTeamSize                   0
WashMaxSize                    26   ( 20.5 %)
washNBuffers                   4     ( 3.1 %)
washNDirtyBuffers              1     (0.8%)

washSignalThreshold            3      ( 2.4 %)
washNActiveSweepers            0
washIntensity                  1

==============================
Memory Manager
==============================
STATS-NAME                   VALUE
MemAllocated                 43616536     (  42594 KB)
MemAllocatedMax              43735080     (  42710 KB)
MemAllocatedEver             0            (      0 KB)
MemNAllocated                67079
MemNAllocatedEver            0
```

```
MemNTimesLocked              0
MemNTimesWaited              0              (    0.0 %)
```

See also

IQ UTILITIES statement in Chapter 1, “SQL Statements,” in *Reference: Statements and Options*

Chapter 5, “Monitoring and Tuning Performance” in the *Performance and Tuning Guide*

sp_iqtable procedure

Function

Displays information about tables in the database.

Syntax1

sp_iqtable ([*table_name*],[*table_owner*],[*table_type*])

Syntax2

sp_iqtable [table_name=*'tablename'*],
[table_owner=*'tableowner'*],[table_type=*'tabletype'*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

Syntax1 If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqtable NULL,NULL,TEMP` and `sp_iqtable NULL,dbo,SYSTEM`.

Note The *table_type* values ALL and VIEW must be enclosed in single quotes in Syntax1.

Syntax2 The parameters can be specified in any order. Enclose them in single quotes.

Table 7-58 lists the allowed values for the *table_type* parameter:

Table 7-58: sp_iqtable table_type values

table_type value	Information displayed
SYSTEM	System tables
TEMP	Global temporary tables
VIEW	Views
ALL	IQ tables, system tables, and views
any other value	IQ tables

Description

Specifying one parameter returns only the tables that match that parameter. Specifying more than one parameter filters the results by all of the parameters specified. Specifying no parameters returns all Sybase IQ tables in the database. There is no method for returning the names of local temporary tables.

Table 7-59: sp_iqtable columns

Column name	Description
table_name	The name of the table.
table_type	BASE – a base table. MAT VIEW - a materialized view. GBL TEMP - a global temporary table. PARTITION - a table partition (this table is for internal use only and cannot be used by Sybase IQ users). VIEW – a view. JVT – a join virtual table.
table_owner	The owner of the table
server_type	IQ – an object created in the IQ store. SA – an object created in the SA store. All views are created in the SA store.
location	TEMP – IQ temporary store. MAIN – IQ store. SYSTEM – catalog store.
dbspace_id	Name of the dbspace where the table resides.
isPartitioned	'Y' if the column belongs to a partitioned table and has one or more partitions whose dbspace is different from the table partition's dbspace, 'N' if the column's table is not partitioned or each partition of the column resides in the same dbspace as the table partition.
remarks	User comments added with the COMMENT statement.
table_constraints	Constraints against the table.

Examples

The following variations in syntax both return information about the table Departments:

```
sp_iqtable ('Departments')
sp_iqtable table_name='Departments'
```

Table_name	Table_type	Table_owner	Server_type	location
Departments	BASE	GROUPO	IQ	Main

dbspace_id	isPartitioned	Remarks	table_constraints
16387	N	(NULL)	(NULL)

The following variations in syntax both return all tables that are owned by table owner GROUPO:

```
sp_iqtable NULL,GROUPO
sp_iqtable table_owner='GROUPO'
```

Table_name	Table_type	Table_owner	Server_type	location
Contacts	BASE	GROUPO	IQ	Main
Customers	BASE	GROUPO	IQ	Main
Departments	BASE	GROUPO	IQ	Main
Employees	BASE	GROUPO	IQ	Main
FinancialCodes	BASE	GROUPO	IQ	Main
FinancialData	BASE	GROUPO	IQ	Main
Products	BASE	GROUPO	IQ	Main
SalesOrders	BASE	GROUPO	IQ	Main
SalesOrderItems	BASE	GROUPO	IQ	Main

dbspace_id	isPartitioned	Remarks	table_constraints
16387	N	(NULL)	(NULL)
16387	N	(NULL)	(NULL)
16387	N	(NULL)	(NULL)
16387	N	(NULL)	(NULL)
16387	N	(NULL)	(NULL)
16387	N	(NULL)	(NULL)
16387	N	(NULL)	(NULL)
16387	N	(NULL)	(NULL)
16387	N	(NULL)	(NULL)

sp_iqtablesize procedure

Function

Returns the size of the specified table.

Syntax

sp_iqtablesize (*table_owner*.*table_name*)

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Returns the total size of the table in KBytes and NBlocks (IQ blocks). Also returns the number of pages required to hold the table in memory, and the number of IQ pages that are compressed when the table is compressed (on disk). You must specify the *table_name* parameter with this procedure. If you are the owner of *table_name*, then you do not have to specify the *table_owner* parameter.

Table 7-60: sp_iqtablesize columns

Column name	Description
Ownername	Name of owner
Tablename	Name of table
Columns	Number of columns in the table
KBytes	Physical table size in KB
Pages	Number of IQ pages needed to hold the table in memory
CompressedPages	Number of IQ pages that are compressed, when the table is compressed (on disk)
NBlocks	Number of IQ blocks

Pages is the total number of IQ pages for the table. The unit of measurement for pages is IQ page size. All in-memory buffers (buffers in the IQ buffer cache) are the same size.

IQ pages on disk are compressed. Each IQ page on disk uses 1 to 16 blocks. If the IQ page size is 128KB, then the IQ block size is 8KB. In this case, an individual on-disk page could be 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, or 128 KB.

If you divide the KBytes value by page size, you see the average on-disk page size.

Note Sybase IQ always reads and writes an entire page, not blocks. For example, if an individual page compresses to 88K, then IQ reads and writes the 88K in one I/O. The average page is compressed by a factor of 2 to 3.

NBlocks is Kbytes divided by IQ block size.

CompressedPages is the number of pages that are compressed. For example, if Pages is 1000 and CompressedPages is 992, this means that 992 of the 1000 pages are compressed. CompressedPages divided by Pages is usually near 100%, because most pages compress. An empty page is not compressed, since Sybase IQ does not write empty pages. IQ pages compress well, regardless of the fullness of the page.

Example

```
call sp_iqtablesize ('dba.emp1')
```

Ownername	Tablename	Columns	KBytes	Pages	CompressedPages	NBlocks
DBA	emp1	4	1504	24	14	188

sp_iqtableuse procedure

Function

Reports detailed usage information for tables accessed by the workload.

Syntax

sp_iqtableuse

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Tables created in SYSTEM are not reported.

Table 7-61: sp_iqtableuse columns

Column name	Description
TableName	Table name
Owner	Username of table owner
UID**	Table unique identifier
LastDT	Date/time of last access
NRef	Number of query references

**UID is a number assigned by the system that uniquely identifies the instance of the table (where instance is defined when an object is created).

See also

“Monitoring workloads,”Chapter 3, “Optimizing Queries and Deletions,” in the *Performance and Tuning Guide*.

“sp_iqcolumnuse procedure” on page 372, “sp_iqdbspace procedure” on page 388, “sp_iqindexadvice procedure” on page 418, “sp_iqindexuse procedure” on page 426, “sp_iqunusedcolumn procedure” on page 481, “sp_iqunusedindex procedure” on page 482, “sp_iqunusedtable procedure” on page 483, and “sp_iqworkmon procedure” on page 490

“INDEX_ADVISOR option” in Chapter 2, “Database Options,” in *Reference: Statements and Options*

sp_iqtransaction procedure

Function

Shows information about transactions and versions.

Syntax

sp_iqtransaction

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

sp_iqtransaction returns a row for each transaction control block in the Sybase IQ transaction manager. The columns Name, Userid, and ConnHandle are the connection properties Name, Userid, and Number, respectively. Rows are ordered by TxnID.

sp_iqtransaction output does not contain rows for connections that do not have a transaction started. To see all connections, use sp_iqconnection.

Note Although you can use sp_iqtransaction to identify users who are blocking other users from writing to a table, sp_iqlocks is a better choice for this purpose.

Table 7-62: sp_iqtransaction columns

Column name	Description
Name	The name of the application.
Userid	The user ID for the connection.
TxnID	The transaction ID of this transaction control block. The transaction ID is assigned during begin transaction. This is the same as the transaction ID displayed in the *.iqmsg* file by the BeginTxn, CmtTxn and PostCmtTxn messages as well as the Txn ID Seq logged when the database is opened.
CmtID	The ID assigned by the transaction manager when the transaction commits. It is zero for active transactions.
VersionID	In simplex databases, the VersionID is the same as the TxnID. For the multiplex coordinator, the VersionID is the same as the TxnID of the active transaction and VersionID is the same as the CmtID of a committed transaction. In multiplex secondary servers, the VersionID is the CmtID of the transaction that created the database version on the multiplex coordinator. It is used internally by the Sybase IQ in-memory catalog and the IQ transaction manager to uniquely identify a database version to all nodes within a multiplex database.
State	The state of the transaction control block. This variable reflects internal Sybase IQ implementation detail and is subject to change in the future. At the time of this writing, transaction states are NONE, ACTIVE, ROLLING_BACK, ROLLED_BACK, COMMITTING, COMMITTED, and APPLIED.
ConnHandle	The ID number of the connection.
IQConnID	The ten-digit connection ID displayed as part of all messages in the *.iqmsg* file. This is a monotonically increasing integer unique within a server session.
MainTableKBCr	The number of kilobytes of IQ store space created by this transaction.
MainTableKBDr	The number of kilobytes of IQ store space dropped by this transaction, but which persist on disk in the store because the space is visible in other database versions or other savepoints of this transaction.
TempTableKBCr	The number of kilobytes of IQ temporary store space created by this transaction for storage of IQ temporary table data.

Column name	Description
TempTableKBDr	The number of kilobytes of IQ temporary table space dropped by this transaction, but which persist on disk in the IQ temporary store because the space is visible to IQ cursors or is owned by other savepoints of this transaction.
TempWorkSpaceKB	For ACTIVE transactions, this is a snapshot of the work space in use at this instant by this transaction, such as sorts, hashes, and temporary bitmaps. The number varies depending on when you run sp_iqtransaction. For example, the query engine might create 60MB in the temporary cache but release most of it quickly, even though query processing continues. If you run sp_iqtransaction after the query finishes, this column shows a much smaller number. When the transaction is no longer active, this column is zero. For ACTIVE transactions, this column is the same as the TempWorkSpaceKB column of sp_iqconnection.
TxnCreateTime	The time the transaction began. All Sybase IQ transactions begin implicitly as soon as an active connection is established or when the previous transaction commits or rolls back.
CursorCount	The number of open Sybase IQ cursors that reference this transaction control block. If the transaction is ACTIVE, it indicates the number of open cursors created within the transaction. If the transaction is COMMITTED, it indicates the number of HOLD cursors that reference a database version owned by this transaction control block.
SpCount	The number of savepoint structures that exist within the transaction control block. Savepoints may be created and released implicitly. Therefore, this number does not indicate the number of user-created savepoints within the transaction.
SpNumber	The active savepoint number of the transaction. This is an implementation detail and might not reflect a user-created savepoint.
MPXServerName	The value indicates if an active transaction is from an inter-node communication (INC) connection. If from INC connection, the value is the name of the multiplex server where the transaction originates. NULL if not from an INC connection. Always NULL if the transaction is not active.
GlobalTxnID	The value indicates the global transaction ID associated with the current transaction. Zero if there is no associated global transaction.

Example

Here is an example of sp_iqtransaction output:

```
Name    Userid  TxnID  CmtID VersionID    State    ConnHandle IQConnID
======= ===== ====== ====== ========= ========== =========== ========
red2      DBA  10058  10700     10058  COMMITTED   419740283       14
blue1     DBA  10568      0     10568     ACTIVE   640038605       17
          DBA  10604      0     10604     ACTIVE  2094200996       18
fromSCJ   DBA  10619      0     10619     ACTIVE   954498130       20
blue2     DBA  10634  10677     10634  COMMITTED   167015670       21
ntJava2   DBA  10676      0     10676     ACTIVE  1779741471       24
blue2     DBA  10678      0     10678     ACTIVE   167015670       21
```

```
nt1       DBA  10699     0    10699    ACTIVE   710225777      28
red2      DBA  10701     0    10701    ACTIVE   419740283      14
          DBA  16687     0    16687    ACTIVE  1306718536      23

MainTableKBCr      MainTableKBDr    TempTableKBCr TempTableKBDr
============= ================== ================================
        0                    0          65824              0
        0                    0              0              0
        0                    0              0              0
        0                    0              0              0
     3960                  152              0              0
        0                    0              0              0
     2400                 1992              0              0
        0                    0              0              0
        0                    0           2912          22096
        0                    0              0              0

TempWorkSpaceKB TxnCreateTime               CursorCount SpCount SpNumber
==============  ==================          ==========  ======  ======
       0    2009-06-26 13:17:27.612                 1       3       2
  102592    2009-06-26 13:27:28.491                 1       1       0
       0    2009-06-26 13:30:27.548                 0       1       0
       0    2009-06-26 13:31:27.151                 0      24     262
       0    2009-06-26 13:35:02.128                 0       0       0
       0    2009-06-26 13:43:58.805                 0      39     408
     128    2009-06-26 13:45:28.379                 0       1       0
       0    2009-06-26 14:05:15.759                 0      42     413
     680    2009-06-26 14:57:51.104                 1       2      20
       0    2009-06-26 15:09:30.319                  0       1       0

MPXServerName  GlobalTxnID
=============  ===========
        (NULL)           0
        (NULL)           0
        (NULL)           0
        (NULL)           0
        (NULL)           0
        (NULL)           0
        (NULL)           0
        (NULL)           0
        (NULL)           0
        (NULL)           0
```

sp_iqunusedcolumn procedure

Function
Reports IQ columns that were not referenced by the workload.

Syntax
sp_iqunusedcolumn

Permissions
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description
Columns from tables created in SYSTEM or local temporary tables are not reported.

Table 7-63: sp_iqunusedcolumn columns

Column name	Description
TableName	Table name
ColumnName	Column name
Owner	Username of column owner

Example
Sample output from the sp_iqunusedcolumn procedure:

```
TableName           ColumnName             Owner
SalesOrders         ID                     GROUPO
SalesOrders         CustomerID             GROUPO
SalesOrders         OrderDate              GROUPO
SalesOrders         FinancialCode          GROUPO
SalesOrders         Region                 GROUPO
SalesOrders         SalesRepresentative    GROUPO
SalesOrderItems     ID                     GROUPO
SalesOrderItems     LineID                 GROUPO
SalesOrderItems     ProductID              GROUPO
SalesOrderItems     Quantity               GROUPO
SalesOrderItems     ShipDate               GROUPO
Contacts            ID                     GROUPO
Contacts            Surname                GROUPO
Contacts            GivenName              GROUPO
...
```

See also
"Monitoring workloads,"Chapter 3, "Optimizing Queries and Deletions," in the *Performance and Tuning Guide*.

"sp_iqcolumnuse procedure" on page 372, "sp_iqdbspace procedure" on page 388, "sp_iqindexadvice procedure" on page 418, "sp_iqindexuse procedure" on page 426, "sp_iqtableuse procedure" on page 477, "sp_iqunusedindex procedure" on page 482, "sp_iqunusedtable procedure" on page 483, and "sp_iqworkmon procedure" on page 490

"INDEX_ADVISOR option" in Chapter 2, "Database Options," in *Reference: Statements and Options*

sp_iqunusedindex procedure

Function

Reports IQ secondary (non-FP) indexes that were not referenced by the workload.

Syntax

sp_iqunusedindex

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Indexes from tables created in SYSTEM or local temporary tables are not reported.

Table 7-64: sp_iqunusedindex columns

Column name	Description
IndexName	Index name
TableName	Table name
Owner	User name of index owner
IndexType	Index type

Example

Sample output from the sp_iqunusedindex procedure:

IndexName	TableName	Owner	IndexType
ASIQ_IDX_T450_I7_HG	SalesOrders	GROUPO	HG
ASIQ_IDX_T450_C6_HG	SalesOrders	GROUPO	HG
ASIQ_IDX_T450_C4_HG	SalesOrders	GROUPO	HG
ASIQ_IDX_T450_C2_HG	SalesOrders	GROUPO	HG
ASIQ_IDX_T451_I6_HG	SalesOrderItems	GROUPO	HG
ASIQ_IDX_T451_C3_HG	SalesOrderItems	GROUPO	HG
ASIQ_IDX_T451_C1_HG	SalesOrderItems	GROUPO	HG
ASIQ_IDX_T452_I11_HG	Contacts	GROUPO	HG
ASIQ_IDX_T453_I10_HG	Contacts	GROUPO	HG
ASIQ_IDX_T454_I4_HG	FinancialCodes	GROUPO	HG
ASIQ_IDX_T455_I5_HG	FinancialData	GROUPO	HG
ASIQ_IDX_T455_C3_HG	FinancialData	GROUPO	HG
ASIQ_IDX_T456_I8_HG	Products	GROUPO	HG
ASIQ_IDX_T457_I4_HG	Departments	GROUPO	HG
ASIQ_IDX_T457_C3_HG	Departments	GROUPO	HG
ASIQ_IDX_T458_I21_HG	Departments	GROUPO	HG
ASIQ_IDX_T458_C5_HG	Departments	GROUPO	HG

See also

“Monitoring workloads,”Chapter 3, “Optimizing Queries and Deletions,” in the *Performance and Tuning Guide*.

“sp_iqcolumnuse procedure” on page 372, “sp_iqdbspace procedure” on page 388, “sp_iqindexadvice procedure” on page 418, “sp_iqindexuse procedure” on page 426, “sp_iqtableuse procedure” on page 477, “sp_iqunusedcolumn procedure” on page 481, “sp_iqunusedtable procedure” on page 483, and “sp_iqworkmon procedure” on page 490

“INDEX_ADVISOR option” in Chapter 2, “Database Options,” in *Reference: Statements and Options*

sp_iqunusedtable procedure

Function

Reports IQ tables that were not referenced by the workload.

Syntax

sp_iqunusedtable

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Tables created in SYSTEM and local temporary tables are not reported.

Table 7-65: sp_iqunusedtable columns

Column name	Description
TableName	Table name
Owner	Username of table owner

Example

The following table illustrates sample output from the sp_iqunusedtable procedure.

```
TableName              Owner
FinancialCodes         GROUPO
Contacts               GROUPO
Employees              GROUPO
emp1                   DBA
SalesOrders            GROUPO
FinancialData          GROUPO
Departments            GROUPO
SalesOrderItems        GROUPO
Products               GROUP
iq_dummy               DBA
Customers              GROUPO
sale                   DBA
```

See also

"Monitoring workloads,"Chapter 3, "Optimizing Queries and Deletions," in the *Performance and Tuning Guide*.

"sp_iqcolumnuse procedure" on page 372, "sp_iqdbspace procedure" on page 388, "sp_iqindexadvice procedure" on page 418, "sp_iqindexuse procedure" on page 426, "sp_iqtableuse procedure" on page 477, "sp_iqunusedcolumn procedure" on page 481, "sp_iqunusedindex procedure" on page 482, and "sp_iqworkmon procedure" on page 490

"INDEX_ADVISOR option" in Chapter 2, "Database Options," in *Reference: Statements and Options*

sp_iqversionuse procedure

Function

Displays version usage for the IQ main store.

Syntax

sp_iqversionuse

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

The sp_iqversionuse system stored procedure helps troubleshoot situations where the databases uses excessive storage space due to multiple table versions.

If out-of-space conditions occur or sp_iqstatus shows a high percentage of main blocks in use on a multiplex server, run sp_iqversionuse to find out which versions are being used and the amount of space that can be recovered by releasing versions. For information on multiplex capability, see *Using Sybase IQ Multiplex*.

The amount of space is expressed as a range because the actual amount typically depends on which other versions are released. The actual amount of space released can be anywhere between the values of MinKBRelease and MaxKBRelease. The oldest version always has MinKBRelease equal to MaxKBRelease.

WasReported indicates whether version usage information has been sent from the secondary server to the coordinator. WasReported is 0 initially on a coordinator for new versions. WasReported changes to 1 once SQL Remote replicates version usage information back to the coordinator. If WasReported is 0 for an extended period, SQL Remote might be stopped.

Note The WasReported column is used in a multiplex setting. For more information on multiplex, see *Using Sybase IQ Multiplex*.

Table 7-66: sp_iqversionuse columns

Column name	Description
VersionID	The version identifier
Server	The server to which users of this version are connected
IQConnID	The connection ID using this version
WasReported	Indicates whether the server has received usage information for this version
MinKBRelease	The minimum amount of space returned once this version is no longer in use
MaxKBRelease	The maximum amount of space returned once this version is no longer in use

Example

The following table illustrates sample output from the sp_iqversionuse system procedure:

```
VersionID Server        IQConnID WasReported
========= ======        ======== ===========
        0 ab2ab_iqdemo        9           0
```

```
MinKBRelease  MaxKBRelease
============  ============
           0             0
```

sp_iqview procedure

Function

Displays information about views in a database.

Syntax1

sp_iqview ([*view_name*],[*view_owner*],[*view_type*])

Syntax2

sp_iqview [view_name=*'viewname'*],
[view_owner=*'viewowner'*],[view_type=*'viewtype'*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

Syntax1 `sp_iqview NULL,NULL,SYSTEM` If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example: `sp_iqview NULL,NULL,SYSTEM` and `sp_iqview deptview,NULL,'ALL'`.

Note The *view_type* value ALL must be enclosed in single quotes in Syntax1.

Syntax2 The parameters can be specified in any order. Enclose them in single quotes.

Table 7-67 lists the allowed values for the *view_type* parameter.

Table 7-67: sp_iqview view_type values

view_type value	Information displayed
SYSTEM	System views
ALL	User and system views
any other value	User views

Description

Specifying one of the parameters returns only the views with the specified view name or views that are owned by the specified user. Specifying more than one parameter filters the results by all of the parameters specified. Specifying no parameters returns all user views in a database.

Table 7-68: sp_iqview columns

Column name	Description
view_name	The name of the view
view_owner	The owner of the view
view_def	The view definition as specified in the CREATE VIEW statement
remarks	User comments added with the COMMENT statement

sp_iqview returns a view definition greater than 32K characters without truncation.

Examples

The following variations in syntax both return information about the view deptview:

```
call sp_iqview('ViewSalesOrders')
sp_iqview view_name='ViewSalesOrders'
```

The following variations in syntax both return all views that are owned by view owner GROUPO:

```
sp_iqview NULL,GROUPO
sp_iqview view_owner='GROUPO'
```

view_name	view_owner	view_def	remarks
ViewSalesOrders	GROUPO	Create views GROUPO , ViewSalesOrders(ID, LineID, ProductID, Quantity, OrderDate, ShipDate, Region, SalesRepresentativeName	(NULL)

sp_iqwho procedure

Function

Displays information about all current users and connections, or about a particular user or connection.

Syntax

sp_iqwho [{ *connhandle* | *user-name* } [, *arg-type*]]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

The sp_iqwho stored procedure displays information about all current users and connections, or about a particular user or connection.

Table 7-69: sp_iqwho columns

Column name	Description
ConnHandle	The SA connection handle
IQConnID	The Sybase IQ specific connection ID
Userid	The name of the user that opened the connection "ConnHandle"
BlockedOn	The connection on which a particular connection is blocked; 0 if not blocked on any connection
BlockUserid	The owner of the blocking connection; NULL if there is no blocking connection
ReqType	The type of the request made through the connection; DO_NOTHING if no command is issued
IQCmdType	The type of Sybase IQ command issued from the connection; NONE if no command is issued
IQIdle	The time in seconds since the last Sybase IQ command was issued through the connection; in case of no last Sybase IQ command, the time since '01-01-2000' is displayed
SAIdle	The time in seconds since the last SA request was issued through the connection; in case of no last SA command, the time since '01-01-2000' is displayed
IQCursors	The number of active cursors in the connection; 0 if no cursors
IQThreads	The number of threads with the connection. At least one thread is started as soon as the connection is opened, so the minimum value for IQThreads is 1.
TempTableSpaceKB	The size of temporary table space in kilobytes; 0 if no temporary table space is used
TempWorkSpaceKB	The size of temporary workspace in kilobytes; 0 if no temporary workspace is used

Adaptive Server Enterprise compatibility The Sybase IQ sp_iqwho stored procedure incorporates the Sybase IQ equivalents of columns displayed by the Adaptive Server Enterprise sp_who procedure. Some Adaptive Server Enterprise columns are omitted, as they are not applicable to Sybase IQ. Table 7-70 maps the Adaptive Server Enterprise sp_who columns to the columns displayed by sp_iqwho.

Table 7-70: Mapping of sp_who and sp_iqwho columns

sp_who column	sp_iqwho column
fid	Family to which a lock belongs; omitted, as not applicable to Sybase IQ
spid	ConnHandle, IQConnID
status	IQIdle, SAIdle
loginame	Userid
origname	User alias; omitted, as not applicable to Sybase IQ
hostname	Name of the host on which the server is running; currently not supported
blk_spid	BlockedOn
dbname	Omitted, as there is one server and one database for Sybase IQ and they are the same for every connection
cmd	ReqType, IQCmdType
block_xloid	BlockUserid

Usage

connhandle An integer representing the connection ID. If this parameter is specified, sp_iqwho returns information only about the specified connection. If the specified connection is not open, no rows are displayed in the output.

user-name A char(255) parameter representing a user login name. If this parameter is specified, sp_iqwho returns information only about the specified user. If the specified user has not opened any connections, no rows are displayed in the output. If the specified user name does not exist in the database, sp_iqwho returns the error message `"User` ***`user-name`*** `does not exist"`

arg-type The *arg-type* parameter is optional and can be specified only when the first parameter has been specified. The only value for *arg-type* is "user". If the *arg-type* value is specified as "user", sp_iqwho interprets the first parameter as a user name, even if the first parameter is numeric. If any value other than "user" is specified for *arg-type*, sp_iqwho returns the error

```
"Invalid parameter."
```

Enclose the *arg-type* value in double quotes.

If no parameters are specified, sp_iqwho displays information about all currently active connections and users.

Either a connection handle or a user name can be specified as the first sp_iqwho parameter. The parameters *connhandle* and *user-name* are exclusive and optional. Only one of these parameters can be specified at a time. By default, if the first parameter is numeric, the parameter is assumed to be a connection handle. If the first parameter is not numeric, it is assumed to be a user name.

Sybase IQ allows numeric user names. The *arg-type* parameter directs sp_iqwho to interpret a numeric value in the first parameter as a user name. For example:

```
sp_iqwho 1, "user"
```

When the *arg-type* "user" is specified, sp_iqwho interprets the first parameter 1as a user name, not as a connection ID. If a user named 1 exists in the database, sp_iqwho displays information about connections opened by user 1.

Table 7-71: sp_iqwho usage examples

Syntax	Output
sp_iqwho	Displays all active connections
sp_iqwho 3	Displays information about connection 3
sp_iqwho "DBA"	Displays connections opened by user DBA
sp_iqwho 3, "user"	Interprets 3 as a user name and displays connections opened by user 3. If user 3 does not exist, returns the error `"User 3 does not exist"`
sp_iqwho non-existing-user	Returns error `"User non-existing-user does not exist"`
sp_iqwho 3, "xyz"	Returns the error `"Invalid parameter: xyz"`

Example

Display all active connections:

```
ConnHandle IQConnID Userid ReqType      IQCmdType            BlockedOn BlockUserid IQCursors
IQThreads IQIdle SAIdle TempTableSpaceKB TempWorkSpaceKB

12          118      DBA    CURSOR_OPEN  IQUTILITYOPENCURSOR  0         (NULL)      0
1         1      0      0              0
13          119      shweta DO_NOTHING   NONE                 0         (NULL)      0
1    16238757   470     0              0
```

See also

"sp_iqconnection procedure" on page 373

"sa_conn_info system procedure" on page 494

sp_iqworkmon procedure

Function

Controls collection of workload monitor usage information, and reports monitoring collection status.

Syntax

sp_iqworkmon [*'action'*] [, *'mode'*]

action = 'start' , 'stop' , 'status' , 'reset'

mode = 'index' , 'table' , 'column' , 'all'

For example:

sp_iqworkmon *'start'* , *'all'*

If one argument is specified, it can only be *action*. For example:

sp_iqworkmon *'stop'*

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

action Specifies the control action to apply. A value of *start* starts monitoring for the specified mode immediately. A value of *stop* stops monitoring immediately. A value of *reset* clears all collected usage information. A value of *status* (the default) displays the current status without changing state.

mode Specifies the type of monitoring to control. The INDEX, TABLE, and COLUMN keywords individually control monitoring of index usage, table usage, and column usage respectively. The default ALL keyword controls monitoring of all usage monitoring features simultaneously.

There is always a result set when you execute sp_iqworkmon. If you specify a specific mode (such as index), only the row for that mode appears.

Usage is collected only for SQL statements containing a FROM clause; for example, SELECT, UPDATE, and DELETE.

Table 7-72: sp_iqworkmon columns

Column name	Description
MonMode	Table, index, or column
Status	Started or stopped
Rowcount	Current number of rows collected

Example

Sample output from the sp_iqworkmon procedure:

```
MonMode     Status      Rowcount
index       started     15
table       started     10
column      started     31
```

See also

"Monitoring workloads,"Chapter 3, "Optimizing Queries and Deletions," in the *Performance and Tuning Guide*.

"sp_iqcolumnuse procedure" on page 372, "sp_iqindexadvice procedure" on page 418, "sp_iqdbspace procedure" on page 388, "sp_iqindexuse procedure" on page 426, "sp_iqtableuse procedure" on page 477, "sp_iqunusedcolumn procedure" on page 481, "sp_iqunusedindex procedure" on page 482, and "sp_iqunusedtable procedure" on page 483

"INDEX_ADVISOR option" in Chapter 2, "Database Options," in *Reference: Statements and Options*

Catalog stored procedures

The following catalog store stored procedures return result sets displaying database server, database, and connection properties in tabular form. These procedures are owned by the dbo user ID. The PUBLIC group has EXECUTE permission on them.

sa_audit_string system procedure

Function — Adds a string to the transaction log.

Syntax — **sa_audit_string** (*string*)

Permissions — DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description — If auditing is turned on, this system procedure adds a comment into the audit log. The string can be a maximum of 200 bytes long.

Example — The following call adds a comment into the audit log:

```
CALL sa_audit_string( 'Auditing test' )
```

sa_checkpoint_execute system procedure

Function — Allows the execution of shell commands during a checkpoint.

Syntax — **sa_checkpoint_execute** '*shell_commands*'

Parameters — **shell_commands** One or more user commands to be executed in a system shell. The shell commands are specific to the system shell. Commands are separated by a semicolon (;).

Permissions — None.

Description — Allows users to execute shell commands to copy a running database from the middle of a checkpoint operation, when the server is quiescent. The copied database can be started and goes through normal recovery, similar to recovery following a system failure.

sa_checkpoint_execute initiates a checkpoint, and then executes a system shell from the middle of the checkpoint, passing the user commands to the shell. The server then waits for the shell to complete, creating an arbitrary size time window during which to copy database files. Most database activity stops while the checkpoint is executing, so the duration of the shell commands should be limited to acceptable user response time.

If the shell commands return a nonzero status, sa_checkpoint_execute returns an error.

Do not use the sa_checkpoint_execute with interactive commands, as the server must wait until the interactive command is killed. Supply override flags to disable prompting for any shell commands that might become interactive; in other words, the COPY, MOVE, and DELETE commands might prompt for confirmation.

The intended use of sa_checkpoint_execute is with disk mirroring, to split mirrored devices.

When using sa_checkpoint_execute to copy *iqdemo.** files to another directory, all files are copied except the *.db* and *.log* files. Error -910 is returned.

This error not a product defect but a Windows limitation; the Windows copy command cannot copy catalog files while they are open by the database.

Example

Assuming you have created a subdirectory named *backup*, the following statement issues a checkpoint, copies all of the iqdemo database files to the backup subdirectory, and completes the checkpoint:

```
sa_checkpoint_execute 'cp iqdemo.* backup/'
```

sa_conn_activity system procedure

Function

Returns the most recently prepared SQL statement for each connection to databases on the server.

Syntax

sa_conn_activity

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Side effects

None

Description

The sa_conn_activity procedure returns a result set consisting of the most recently prepared SQL statement for each connection if the server has been told to collect the information. To obtain the result set, specify the -zl option when starting the database server or execute the following:

```
CALL sa_server_option('Remember_last_statement','ON')
```

This procedure is useful when the database server is busy and you want to obtain information about what SQL statement is prepared for each connection. This feature can be used as an alternative to request-level logging.

For information on the LastStatement property from which these values are derived, see *SQL Anywhere Server – Database Administration*.

For information about the -zl command line option, see Chapter 1, "Running the Database Server" in *Utility Guide*.

For information about the remember_last_statement setting, see "sa_server_option system procedure" on page 504.

sa_conn_info system procedure

Function

Reports connection property information.

Syntax

sa_conn_info ([*connection-id*])

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Returns a result set consisting of the following connection properties for the supplied connection. If no *connection-id* is supplied, information for all current connections to databases on the server is returned.

- Number
- Name
- Userid
- DBNumber
- LastReqTime
- ReqType
- CommLink
- NodeAddr
- ClientPort
- ServerPort
- BlockedOn
- LockRowID

- LockIndexID
- LockTable
- UncommitOps

In a deadlock situation, the BlockedOn value returned by this procedure allows you to check which users are blocked, and who they are blocked on.

Example

```
sa_conn_info
```

```
569851433,'','DBA',0,'','0.0',1,
'CURSOR_OPEN','local','',6821,0,0,1008
```

sa_conn_properties system procedure

Function

Reports connection property information.

Syntax

sa_conn_properties ([*connection-id*])

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Returns the connection ID as Number, PropNum, PropName, PropDescription, and Value for each available connection property. Omitting the *connection-id* produces results for all connections.

For a listing of available connection properties, see “Connection properties” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Configuring Your Database > Connection, database, and database server properties*

Example

```
sa_conn_properties
```

```
Number    PropNum PropName   PropDescription   Value
1,2,'BytesReceived','Bytes received by server','4157'
1,3,'BytesReceivedUncomp','Bytes received after decompression','4241'
1,4,'BytesSent','Bytes sent to client','15192'
1,10,'CacheHits','Cache Hits','1291'
1,11,'CacheReadIndInt','Cache index interior reads','58'
1,12,'CacheReadIndLeaf','Cache index leaf reads','121'
1,15,'CacheRead','Cache reads','1318'
1,19,'CacheReadTable','Cache table reads','387'
1,20,'CacheReadWorkTable','Cache work table reads','4'
1,21,'CarverHeapPages','Cache pages used for carvers','0'
1,39,'ClientStmtCacheHits','Number of prepares not required because of
```

```
the client statement cache','0'
1,40,'ClientStmtCacheMisses','Number of prepares in the client
statement cache which were prepared again','0'
1,41,'Commit','Number of commit requests','1'
1,48,'Cursor','Declared cursors','4'1,41,'Commit','Number of commit
requests','1'
```

Note To get cache hit statistics for the entire cache, use sa_eng_properties, and see the output lines for CacheHitsEng, CacheReadEng, and DiskReadEng. If you run the same query on the catalog store repeatedly, the first time you should see reads increase but no cache hits; as you repeat the query, cache hits increase in step with cache reads.

sa_db_info system procedure

Function

Reports database property information.

Syntax

sa_db_info ([*database-id*])

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Returns a single row containing the Number, Alias, File, ConnCount, PageSize, and LogName for the specified database.

Example

- The following statement returns a single row describing the current database. Table 7-73 lists sample values.

```
sa_db_info
```

```
0,'iqdemo','
/sys1/users/test/sybase/IQ-15_1/demo/iqdemo.db',
1,4096,'/sys1/users/test/sybase/IQ-15_1/demo/
iqdemo.log'
```

Table 7-73: sa_db_info sample values

Property	Value
Number	0
Alias	iqdemo
File	C:\Documents and Settings\All Users\SybaseIQ\demo\iqdemo.db
ConnCount	1
PageSize	4096

Property	Value
LogName	C:\Documents and Settings\All Users\SybaseIQ\demo\iqdemo.log

See also
"sa_db_properties system procedure" on page 497

sa_db_properties system procedure

Function

Reports database property information.

Syntax

sa_db_properties ([*database-id*])

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Returns the database ID number and the Number, PropNum, PropName, PropDescription, and Value, for each property returned by the sa_db_info system procedure.

Example

```
sa_db_properties
```

```
0,10,'CacheHits','Cache Hits','4660'
0,11,'CacheReadIndInt','Cache index interior reads','334'
0,12,'CacheReadIndLeaf','Cache index leaf reads','1117'
0,15,'CacheRead','Cache reads','4887'
0,19,'CacheReadTable','Cache table reads','2025'
0,20,'CacheReadWorkTable','Cache work table reads','4'
0,22,'ChkptFlush','Checkpoint flushed pages','64'/
0,23,'ChkptPage','Checkpoint log page images saved','64'
0,24,'CheckpointUrgency','Checkpoint Urgency','7'
0,25,'Chkpt','Checkpoints','2'
0,26,'CheckpointLogBitmapSize','Checkpoint log bitmap
size','0'
'0,27,'CheckpointLogBitmapPagesWritten','Checkpoint log
writes to bitmap','0'
0,28,'CheckpointLogCommitToDisk','Checkpoint log commit to
disk','8
0,29,'CheckpointLogPageInUse','Checkpoint log pages in
use','62'
0,30,'CheckpointLogPagesRelocated','Checkpoint log pages
relocated','0'
0,31,'CheckpointLogSavePreimage','Checkpoint log save
preimage','64
```

See also
"sa_db_info system procedure" on page 496

sa_enable_auditing_type system procedure

Function
: Enables auditing and specifies which events to audit.

Syntax
: **sa_enable_auditing_type**(['*string*])

Parameters
: *string* is a comma-delimited string.

Permissions
: DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description
: sa_enable_auditing_type works with the PUBLIC.AUDITING option to enable auditing of specific types of information.

If you set the PUBLIC.AUDITING option to ON, and do not specify which type of information to audit, the default setting (all) takes effect. In this case, all types of auditing information are recorded.

If you set the PUBLIC.AUDITING option to ON, and disable all types of auditing using sa_disable_auditing_type, no auditing information is recorded. To re-establish auditing, use sa_enable_auditing_type to specify which type of information you want to audit.

If you set the PUBLIC.AUDITING option to OFF, then no auditing information is recorded, regardless of the sa_enable_auditing_type setting.

Example
: To enable only option auditing:

```
sa_disable_auditing_type('all')
```

```
sa_enable_auditing_type('options')
```

See also
: AUDITING option [database] in Chapter 2, “Database Options,” in *Reference: Statements and Options*

sa_eng_properties system procedure

Function
: Reports database server property information.

Syntax
: **sa_eng_properties**

Permissions
: DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description
: Returns the PropNum, PropName, PropDescription, and Value for each available server property.

Table 7-74: sa_eng_properties result sets

Column name	Data type	Description
PropNum	integer	The database server property number
PropName	varchar(255)	The database server property name
PropDescription	varchar(255)	The database server property description
Value	long varchar	The database server property value

For a listing of available database server properties, see "System functions" in *SQL Anywhere Server – SQL Reference*. See also "Server-level properties" in *SQL Anywhere Server – Database Administration*.

Examples

The following statement returns a set of available server properties:

```
call sa_eng_properties()
```

PropNum	PropName	...
1	IdleWrite	...
2	IdleChkPt	...
...	...	...

The following statement returns a set of available server properties:

```
sa_eng_properties
```

```
0,'ActiveReq','Active requests','1'
1,'AvailIO','Number of available I/O control blocks','255'
2,'BytesReceived','Bytes received by server','85898'
3,'BytesReceivedUncomp','Bytes received after
decompression','85898'
4,'BytesSent','Bytes sent to client','145053'
5,'BytesSentUncomp','Bytes sent before compression','145053'
6,'CacheAllocated','Cache pages that have been allocated for
server data structures','546'
7,'CacheFile','Cache pages used to hold data from database
files','600'
8,'CacheFileDirty','Cache pages that are dirty (needing a
write)','2'
9,'CacheFree','Number of cache pages not being used','270'
13,'CachePanics','Number of times the cache manager has failed
to find a page to allocate','0'
14,'CachePinned','Pinned cache pages','591'
15,'CacheRead','Cache reads','456801'
16,'CacheReplacements','Cache replacements','0'
17,'CacheScavengeVisited','Number of pages visited while
scavenging for a page to allocate','1416'
18,'CacheScavenges','Number of times the cache manager has
```

```
scavenged for a page to allocate','1416'
21,'CarverHeapPages','Cache pages used for carvers','0'
39,'ClientStmtCacheHits','Number of prepares not required
because of the client statement cache','23'
40,'ClientStmtCacheMisses','Number of prepares in the client
statement cache which were prepared again','3'
44,'CurrentCacheSize','Current cache size in
kilobytes','49152'
51,'DiskRead','Disk reads','938'
54,'DiskReadHintScatterLimit','Imposed limit on the size (in
bytes) of a scatter read hint','0'
55,'DiskRetryRead','Disk read retries','0'
```

sa_table_page_usage system procedure

Function

Reports information about the usage of database tables.

Syntax

sa_table_page_usage

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

The results include the same information provided by the Information utility.

sa_disable_auditing_type system procedure

Function

Disables auditing of specific events.

Syntax

sa_disable_auditing_type(['*string*])

Parameters

string is a comma-delimited string containing one or more of:

all enables all types of auditing.

connect enables auditing of both successful and failed connection attempts.

connectFailed enables auditing of failed connection attempts.

DDL enables auditing of DDL statements.

options enables auditing of public options.

permission enables auditing of permission checks, user checks, and setuser statements.

permissionDenied enables auditing of failed permission and user checks.

triggers enables auditing after a trigger event.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

You can use the sa_disable_auditing_type system procedure to disable auditing of one or more categories of information.

Setting this option to all disables all auditing. You can also disable auditing by setting the public.auditing option to OFF.

sa_flush_cache system procedure

Function

Empties all pages in the database server cache.

Syntax

sa_flush_cache

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Database administrators can use this procedure to empty the contents of the database server cache. This procedure affects the catalog store. It is of use in performance measurement to ensure repeatable results.

sa_make_object system procedure

Function

Used in a SQL script, ensures that a skeletal instance of an object exists before executing an ALTER statement that provides the actual definition.

Syntax

sa_make_object (*objtype*, *objname* [, *owner* [, *tabname*])

object-type: ‘**procedure**’ | ‘**function**’ | ‘**view**’ | ‘**trigger**’

Permissions

Resource authority required to create or modify database objects.

Description

This procedure is particularly useful in scripts or command files that are run repeatedly to create or modify a database schema. A common problem in such scripts is that the first time they are run, a CREATE statement must be executed, but subsequent times an ALTER statement must be executed. This procedure avoids the necessity of querying the system tables to find out whether the object exists.

To use the procedure, follow it by an ALTER statement that contains the entire object definition.

You can also use the sa_make_object system procedure to add a skeleton Web service.

```
CALL sa_make_object( 'service','my_web_service' )
```

Table 7-75 lists the meaning of the sa_make_object parameters.

Table 7-75: sa_make_object options

Option name	Values
objtype	The type of object being created. The parameter must be one of 'procedure', 'function', 'view', 'service', or 'trigger'.
objname	The name of the object to be created.
owner	The owner of the object to be created. The default value is CURRENT USER.
tabname	Required only if objtype is 'trigger', in which case it specifies the name of the table on which the trigger is to be created.

Examples

- The following statements ensure that a skeleton procedure definition is created, define the procedure, and grant permissions on it. A command file containing these instructions can be run repeatedly against a database without error.

```
CALL sa_make_object( 'procedure','myproc' );
ALTER PROCEDURE myproc( in p1 int, in p2 char(30) )
BEGIN
    // ...
END;
GRANT EXECUTE ON myproc TO public;
```

- The following example uses the sa_make_object system procedure to add a skeleton Web service.

```
CALL sa_make_object( 'service','my_web_service' )
```

See also

"sa_db_info system procedure" on page 496

sa_rowgenerator system procedure

Function

Returns a result set with rows between a specified start and end value.

Syntax

sa_rowgenerator ([*rstart* [, *rend* [, *rstep*]] **]**)

Parameters

- ***rstart*** This optional integer parameter specifies the starting value. The default value is 0.

- ***rend*** This optional integer parameter specifies the ending value. The default value is 100.
- ***rstep*** This optional integer parameter specifies the increment by which the sequence values are increased. The default value is 1.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Result sets

Column name	Data type	Description
row_num	integer	Sequence number.

Remarks

The sa_rowgenerator procedure can be used in the FROM clause of a query to generate a sequence of numbers. This procedure is an alternative to using the RowGenerator system table. You can use sa_rowgenerator for such tasks as:

- Generating test data for a known number of rows in a result set.
- Generating a result set with rows for values in every range. For example, you can generate a row for every day of the month, or you can generate ranges of PostalCodes.
- Generating a query that has a specified number of rows in the result set. This may be useful for testing the performance of queries.

You can emulate the behavior of the RowGenerator table with the following statement:

```
SELECT row_num FROM sa_rowgenerator(1255)
```

Side effects

None

Example

The following query returns a result set containing one row for each day of the current month:

```
SELECT dateadd(day,row_num-1,
ymd(datepart(year,CURRENT DATE),
datepart(month,CURRENT DATE), 1)) AS
day_of_month FROM sa_RowGenerator(1,31,1) WHERE
datepart(month,day_of_month) =
datepart(month,CURRENT DATE) ORDER BY row_num
```

The following query shows how many employees live in zip code ranges (0-9999), (10000-19999), ..., (90000-99999). Some of these ranges have no employees, which causes the warning `Null value eliminated in aggregate function (-109)`. The sa_rowgenerator procedure can be used to generate these ranges, even though no employees have a PostalCode in the range.

```
SELECT row_num AS r1, row_num+9999 AS r2,
count(PostalCode) AS zips_in_range
FROM sa_rowgenerator(0,99999,10000) D LEFT JOIN
employees ON PostalCode
BETWEEN r1 AND r2 GROUP BY r1, r2 ORDER BY 1
```

The following example generates 10 rows of data and inserts them into the emp table:

```
INSERT INTO emp1(dept_id, salary, name) SELECT row_num,
CAST( rand() * 1000 AS INTEGER), 'Mary' FROM
sa_rowgenerator(1, 10)
```

sa_server_option system procedure

Function

Overrides a database server command line option while the database server is running.

Syntax

sa_server_option (*option_name*, *option_value*)

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Database administrators can use this procedure to override some database server options without restarting the database server.

You can reset these options:

Option name	Values	Default
Disable_connections	ON or OFF	OFF
IQMsgMaxSize	Integer 0 – 2047 (inclusive) in megabytes	0
IQMsgNumFiles	Integer 0 – 64 (inclusive)	0
Liveness_timeout	Integer, in seconds	120
Main_Cache_Cemory_MB	1 – 4294967295 (2^{32} - 1)	32MB
Temp_Cache_Memory_MB	1 – 4294967295 (2^{32} - 1)	24MB
Procedure_profiling	ON, OFF, RESET, CLEAR	OFF
Profile_filter_conn	connection-id	
Profile_filter_user	user-id	
Quitting_time	Valid date and time	

Option name	Values	Default
Remember_last_statement	ON or OFF	OFF
Request_level_log_file	*Filename*	
Request_level_log_size	*File-size*, in bytes,	
Request_level_logging	ALL, SQL, NONE, SQL+hostvars	NONE
Requests_for_connection	connection-id, -1	
Requests_for_database	database-id, -1	

disable_connections When set to ON, no other connections are allowed to any databases on the database server.

IQMsgMaxSize Limits the maximum size of the message log. IQMsgMaxSize is an integer 0-2047 (inclusive), in megabytes. The default value is 0, which specifies that there is no limit on message log size. IQMsgMaxSize corresponds to the -iqmsgsz server switch and takes precedence over the value of -iqmsgsz.

A IQMsgMaxSize value *n* greater than 0 means that the message log can grow up to *n* megabytes in size. For example, the following statement limits the size of the message log to 50MB:

```
CALL sa_server_option('IQMsgMaxSize','50')
```

A -iqmsgsz value *n* greater than 0 means that the message log can grow up to *n* megabytes in size. For example, the following command limits the size of the message log to 100MB:

```
start_iq -n iqdemo iqdemo.db ... <other options> ...
-iqmsgsz 100
```

For information on the -iqmsgsz server startup switch, see “Starting the database server” in Chapter 1, “Running the Database Server” of the *Utility Guide*.

For information on message log management, see “Message logging” in Chapter 1, “Overview of Sybase IQ System Administration” of the *System Administration Guide: Volume 1*.

IQMsgNumFiles Specifies the number of archives of the old message log maintained by the server. The value of IQMsgNumFiles takes effect only if the IQMsgMaxSize server property or the -iqmsgsz server startup switch is non-zero. IQMsgNumFiles corresponds to the -iqmsgnum server switch and takes precedence over the value of -iqmsgnum.

IQMsgNumFiles is an integer 0-64 (inclusive). The default value is 0, which means that messages are wrapped in the main message log file.

A IQMsgNumFiles value *n* greater than 0 means that the server maintains *n* message log archives. For example, the following statement specifies that the server maintain 5 archives of the message log:

```
CALL sa_server_option('IQMsgNumFiles','5')
```

For information on the -iqmsgnum server startup switch, see "Starting the database server" in Chapter 1, "Running the Database Server" of the *Utility Guide*.

For information on message log management, see "Message logging" in Chapter 1, "Overview of Sybase IQ System Administration" of the *System Administration Guide: Volume 1*.

liveness_timeout A liveness packet is sent periodically across a client/server TCP/IP or SPX network to confirm that a connection is intact. If the network server runs for a liveness_timeout period without detecting a liveness packet, the communication is severed.

For information on the -tl command line option, see "Limiting inactive connections" in Chapter 2, "Running Sybase IQ" in *System Administration Guide: Volume 1*.

main_cache_memory_mb Changes the default of the main shared memory buffer cache dynamically. This option can be set on a running server but cannot change cache size on a running database. If two databases need to run with different cache sizes, set the option before starting each database. If the cache size is set using the -iqmc server startup switch, Sybase IQ uses the specified value for all databases started on that server unless sa_server_option specifies a new value. For more information, see "Buffer caches and physical memory" in Chapter 4, "Managing System Resources," in the *Performance and Tuning Guide*.

procedure_profiling Controls procedure profiling for stored procedures, functions, events, and triggers. Procedure profiling shows you how long it takes your stored procedures, functions, events, and triggers to execute, as well as how long each line takes to execute. You can also set procedure profiling options on the Database property sheet in Sybase Central. Collected information appears on the Profile tab in the right pane of Sybase Central.

- **ON** enables procedure profiling for the database you are currently connected to.
- **OFF** disables procedure profiling and leaves the profiling data available for viewing.
- **RESET** returns the profiling counters to zero, without changing the ON or OFF setting.

- **CLEAR** returns the profiling counters to zero and disables procedure profiling.

Once profiling is enabled, you can use the sa_procedure_profile_summary and sa_procedure_profile stored procedures to retrieve profiling information from the database. For more information about these procedures, see *SQL Anywhere Server – SQL Reference*.

For more information about viewing procedure profiling information in Sybase Central, see “Profiling database procedures” in the *Performance and Tuning Guide*.

profile_filter_conn Instructs the database server to capture profiling information for a specific connection ID.

profile_filter_user Instructs the database server to capture profiling information for a specific user ID.

quitting_time Instructs the database server to shut down at the specified time.

For more information on quitting_time, see the -tq server option in Chapter 1, “Running the Database Server” in the *Utility Guide*.

remember_last_statement Instructs the database server to capture the most recently prepared SQL statement for each connection to databases on the server. For stored procedure calls, only the outermost procedure call appears, not the statements within the procedure.

You can obtain the current value of the remember_last_statement setting using the RememberLastStatement property function as follows:

```
SELECT property( 'RememberLastStatement' )
```

For more information, see -zl server option in Chapter 1, “Running the Database Server” in the *Utility Guide.*

When remember_last_statement is turned on, the following statement returns the most recently prepared statement for the specified connection.

```
SELECT connection_property( 'LastStatement', conn_id )
```

The stored procedure sa_conn_activity returns this same information for all connections.

request_level_log_file The name of the file used to record logging information. A name of NULL stops logging to file. Any backslash characters in the file name must be doubled, as this is a SQL string.

request_level_log_size The maximum size of the file used to record logging information, in bytes.

When the request-level log file reaches the size specified by either the sa_server_option system procedure or the -zs server option, the file is renamed with the extension *.old* appended (replacing an existing file with the same name if one exists). The request-level log file is then restarted.

request_level_logging The logging options can be ALL, SQL, NONE, HOSTVARS, PLAN, PROCEDURES, and TRIGGERS, separated by "+". ON and ALL are equivalent. OFF and NONE are equivalent. This call turns on logging of individual SQL statements sent to the database server, for use in troubleshooting, in conjunction with the database server -zr and -zo options.

SQL includes basic SQL statement related requests. ALL includes SQL requests, plus other requests, which can significantly increase the size of the log. ALL also enables recording of host variable values. If TRIGGERS is specified, all stored procedure statements (including those in triggers) are recorded in the request log.

The settings request_level_debugging and request_level_logging are equivalent.

When you set request_level_logging to OFF, the request-level log file is closed.

If you select SQL, only the following types of request are recorded:

- START DATABASE
- STOP ENGINE
- STOP DATABASE
- Statement preparation
- Statement execution
- EXECUTE IMMEDIATE statements
- Option settings
- COMMIT statements
- ROLLBACK statements
- PREPARE TO COMMIT operations
- Connections
- Disconnections
- Beginnings of transactions
- DROP STATEMENT statement
- Cursor explanations

- Cursor closings
- Cursor resume
- Errors

Setting request_level_logging to SQL+HOSTVARS outputs *both* SQL (as though you specified `request_level_logging=SQL`) *and* host variable values to the log.

You can find the current value of the request_level_logging setting using `property('RequestLogging').`

For more information, see the -z, -zr, -zs, -zo, and -o command line options in Chapter 1, "Running the Database Server" in the *Utility Guide*.

See "-zr level" on page 30 in the *Utility Guide* for a list of requests that are logged by SQL request-level logging. See "Logging server requests" in Chapter 14, "Troubleshooting Hints" of *System Administration Guide: Volume 1* for more information on using request logging. See also "Request logging" in *SQL Anywhere Server – SQL Usage > Monitoring and Improving Database Performance > Improving database performance > Other diagnostic tools and techniques*.

requests_for_connection Filter the request-level logging information so that only information for a particular connection is logged. This can help reduce the size of the request-level log file when monitoring a server with many active connections or multiple databases. You can obtain the connection ID by executing the following:

```
CALL sa_conn_info()
```

To specify a specific connection to be logged once you have obtained the connection ID, execute the following:

```
CALL sa_server_option( 'requests_for_connection',
connection-id )
```

Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:

```
CALL sa_server_option( 'requests_for_connection', -1)
```

requests_for_database Filter the request-level logging information so that only information for a particular database is logged. This can help reduce the size of the request-level log file when monitoring a server with many active connections or multiple databases. You can obtain the database ID by executing the following statement when you are connected to the desired database:

```
SELECT connection_property( 'DBNumber' )
```

To specify that only information for a particular database is to be logged, execute the following:

```
CALL sa_server_option( 'requests_for_database',
database-id )
```

Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:

```
CALL sa_server_option( 'requests_for_database', -1 )
```

temp_cache_memory_mb Changes the default size of the temporary shared memory buffer cache dynamically. This option can be set on a running server but cannot change cache size on a running database. If two databases need to run with different cache sizes, set the option before starting each database. If the cache size is set using the -iqtc server startup switch, Sybase IQ uses the specified value for all databases started on that server unless sa_server_option specifies a new value. For more information, see "Buffer caches and physical memory" in Chapter 4, "Managing System Resources," in the *Performance and Tuning Guide*.

Examples

The following statement disallows new connections to the database server.

```
call sa_server_option( 'disable_connections', 'ON')
```

The following statement changes the size of the main shared memory buffer cache:

```
call sa_server_option( 'main_cache_memory_mb', '200')
```

You must restart the database for the new size to take effect.

See also

"sa_get_request_profile system procedure," "sa_get_request_times system procedure," and "sa_statement_text system procedure" in *SQL Anywhere Server – SQL Reference*.

sa_set_http_header system procedure

Function

Permits a Web service to set an HTTP header in the result.

Syntax

sa_set_http_header (*field-name*, *value*)

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

```
call dbo.sa_set_http_header( 'Content-Type', 'text/
html' )
```

Setting the special header field @HttpStatus sets the status code returned with the request. For example, the following command sets the status code to 404 Not Found.

```
dbo.sa_set_http_header( '@HttpStatus', '404' )
```

The body of the error message is inserted automatically. Only valid HTTP error codes can be used. Setting the status to an invalid code causes an SQL error.

See also

"sa_set_http_option system procedure" on page 511

sa_set_http_option system procedure

Function

Permits a Web service to set an HTTP option in the result.

Syntax

sa_set_http_option (*option-name*, *value*)

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Use this procedure within statements or procedures that handle Web services to set options within an HTTP result set.

Currently only one option is supported:

- **CharsetConversion** Controls whether the result set is to be automatically converted from the character set of the database to the character set of the client. The only permitted values are ON and OFF. The default value is ON.

See also

"sa_set_http_header system procedure" on page 510

sa_validate system procedure

Function

Validates all tables in the catalog store.

Syntax

sa_validate [*tbl_name*,] [*owner_name*,] [*check_type*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

This procedure validates each SQL Anywhere table or index in the catalog store.

For more information, see "Validation utility (dbvalid)" in Chapter 3, "Database Administration Utilities" in the *Utility Guide*.

Table 7-76 lists the meaning of the sa_validate parameters.

Table 7-76: sa_validate options

Option name	Values
tbl_name	Validate only the specified table. When NULL (the default), validate all tables.
owner_name	Validate only the tables owned by the specified user. When NULL (the default), validate tables for all users.
check_type	When NULL (the default), each table is checked without additional checks. The *check_type* value can be one of the following: data, express, full, index, or checksum.

Values for the *tbl_name*, *owner_name*, and *check_type* parameters are strings and must be enclosed in quotes.

The procedure returns a single column, named Messages. If all tables are valid, the column contains:

```
No errors detected
```

Warning! Validate a table or the entire catalog store while no connections are making changes to the database; otherwise, spurious errors might be reported, indicating some form of database corruption even though no corruption actually exists.

Example

The following statement validates all of the catalog store tables with an index check owned by the DBA:

```
CALL sa_validate (owner_name='DBA', check_type =
'index')
```

sa_verify_password system procedure

Function

Validates the password of the current user.

Syntax

sa_verify_password (*string* **)**

Parameters

- ***string*** This char(128) parameter specifies the password of the current database user.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Remarks

This procedure is used by sp_password. If the password matches, the procedure simply returns. If it does not match, the error string returned by the procedure is returned.

Side effects

None.

sp_login_environment system procedure

The sp_login_environment system procedure is a SQL Anywhere system procedure. See "sp_login_environment system procedure" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > System procedures*.

sp_remote_columns system procedure

Function

Produces a list of the columns on a remote table, and a description of those columns. For each column, the procedure returns its database, owner, table, column, domain ID, width, scale, and nullability.

The server must be defined with the CREATE SERVER statement to use this system procedure.

Note You cannot capture output from this procedure in a file. If you use the redirection operator, you receive the message "Cursor is restricted to Fetch Next operations."

Syntax

sp_remote_columns *servername* [, *tablename*] [, *owner*] [, *database*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

If you are entering a CREATE EXISTING statement and you are specifying a column list, it might be helpful to get a list of the columns that are available on a remote table. sp_remote_columns produces a list of the columns on a remote table and a description of those data types.

Example

Gets a list of the columns in the sysobjects table in the production database in an Adaptive Server server named asetest:

```
sp_remote_columns asetest, sysobjects,
null, production
```

Standards and compatibility

Sybase Supported by Open Client/Open Server.

See also

Chapter 5, "Server Classes for Remote Data Access" and Chapter 4, "Accessing Remote Data" in the *System Administration Guide: Volume 2*

CREATE SERVER statement in *Reference: Statements and Options*

sp_remote_exported_keys system procedure

Function

Provides information about tables with foreign keys on a specified primary key table.

The server must be defined with the CREATE SERVER statement to use this system procedure.

Syntax

sp_remote_exported_keys *@server_name* , *@sp_name*
[, *@sp_owner*] [, *@sp_qualifier*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

The sp_remote_exported_keys result set includes the database, owner, table, column, and name for both the primary and the foreign key, as well as the foreign-key sequence for the foreign-key column. The result set might vary because of the underlying ODBC and JDBC calls, but information about the table and column for a foreign key is always returned.

To use sp_remote_exported_keys, your database must be created or upgraded using version 12.4.3 or higher of Sybase IQ.

Parameters

Table 7-77 lists the sp_remote_exported_keys parameters.

Table 7-77: sp_remote_exported_keys parameters

Name	Data type	Description
@server_name	varchar	Server on which the primary-key table is located. Required.
@sp_name	varchar(30)	Table containing the primary key. Required.
@sp_owner	varchar	Owner of primary-key table. Optional.
@sp_qualifier	varchar	Database containing the primary-key table. Optional.

Example

To get information about the remote tables with foreign keys on the sysobjects table, in the production database, in a server named asetest:

```
call sp_remote_exported_keys
```

```
(@server_name='asetest', @sp_name='sysobjects',
@sp_qualifier='production')
```

See also

Chapter 4, "Accessing Remote Data" and Chapter 5, "Server Classes for Remote Data Access" in the *System Administration Guide: Volume 2*

CREATE SERVER statement in *Reference: Statements and Options*

sp_remote_imported_keys system procedure

Function

Provides information about remote tables with primary keys that correspond to a specified foreign key.

The server must be defined with the CREATE SERVER statement to use this system procedure.

Syntax

sp_remote_imported_keys *@server_name* , *@sp_name* [, *@sp_owner*] [, *@sp_qualifier*]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Foreign keys reference a row in a separate table that contains the corresponding primary key. This procedure allows you to obtain a list of the remote tables with primary keys that correspond to a particular foreign key table. The sp_remote_imported_keys result set includes the database, owner, table, column, and name for both the primary and the foreign key, as well as the foreign key sequence for the foreign key column. The result set might vary because of the underlying ODBC and JDBC calls, but information about the table and column for a primary key is always returned.

To use sp_remote_exported_keys, your database must be created or upgraded using version 12.4.3 or higher of Sybase IQ.

Parameters

Table 7-78 lists the sp_remote_imported_keys parameters.

Table 7-78: sp_remote_imported_keys parameters

Name	Data type	Description
@server_name	varchar	Server on which the foreign-key table is located. Required.
@sp_name	varchar(30)	Table containing the foreign key. Required.
@sp_owner	varchar	Owner of foreign-key table. Optional.
@sp_qualifier	varchar	Database containing the foreign-key table. Optional.

Example

Gets information about the tables with primary keys that correspond to a foreign key on the sysobjects table, owned by "fred", in the asetest server:

```
call sp_remote_imported_keys
(@server_name='asetest', @sp_name='sysobjects',
@sp_qualifier='production')
```

See also

Chapter 4, "Accessing Remote Data" and Chapter 5, "Server Classes for Remote Data Access" in the *System Administration Guide: Volume 2*.

CREATE SERVER statement in *Reference: Statements and Options*

sp_remote_primary_keys system procedure

Function

Provides primary key information about remote tables using remote data access.

Syntax

sp_remote_primary_keys *@server_name* [, *@table_name*]
[, *@table_owner*] [, *@table_qualifier*]

Accepts these parameters:

@server_name Selects the server on which the remote table is located.

@table_name Selects the remote table.

@table_owner Selects the owner of the remote table.

@table_qualifier Selects the database.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Side effects

None

Description

Because of differences in the underlying ODBC/JDBC calls, the information returned differs slightly in terms of the catalog/database value, depending upon the remote data access class that is specified for the server. However, the important information (for example, column name) is as expected.

Standards and compatibility

Sybase Supported by Open Client/Open Server.

sp_remote_tables system procedure

Function

Returns a list of the tables on a server.

Syntax

sp_remote_tables *servername* [, *tablename*] [, *owner*]
[, *table_qualifier*] [, *with_table_type*]

The server must be defined with the CREATE SERVER statement to use this system procedure.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

It might be helpful when configuring your database server to get a list of the remote tables available on a particular server. sp_remote_tables returns a list of the tables on a server.

The procedure accepts five parameters:

server_name Selects the server the remote table is located on.

table_name Selects the remote table.

table_owner Selects the owner of the remote table.

table_qualifier Selects the database.

with_table_type Selects the type of remote table. This parameter is a bit type and accepts two values, 0 (the default) and 1. You must enter the value 1 if you want the result set to include a column that lists table types.

The with_table_type parameter is available only for databases created in SQL Anywhere 7.0.2 and higher. If you use this parameter with an older database, the following error message is returned:

```
Wrong number of parameters to function 'sp_remote_tables'
```

If a table, owner, or database name is given, the list of tables is limited to only those that match the parameters.

Note You cannot capture output from this procedure in a file. If you use the redirection operator, you receive the message “Cursor is restricted to Fetch Next operations.”

Examples

- Lists all the Microsoft Excel worksheets available from an ODBC data source named “exce”:

  ```
  sp_remote_tables  excel
  ```

- Lists all the tables in the production database in an Adaptive Server Enterprise server named asetest, owned by user fred:

  ```
  sp_remote_tables  asetest, null, fred, production
  ```

Standards and compatibility

Sybase Supported by Open Client/Open Server.

See also

CREATE SERVER statement in *Reference: Statements and Options*

Chapter 4, “Accessing Remote Data” and Chapter 5, “Server Classes for Remote Data Access” in the *System Administration Guide: Volume 2*

sp_servercaps system procedure

Function

Displays information about a remote server’s capabilities.

The server must be defined with the CREATE SERVER statement to use this system procedure.

Syntax

sp_servercaps *servername*

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Sybase IQ uses capability information to determine how much of a SQL statement can be forwarded to a remote server. The system tables that contain server capabilities are not populated until after Sybase IQ connects to the remote server. This information comes from syscapability and syscapabilityname system tables. The *servername* specified must be the same server name used in the CREATE SERVER statement.

Example

Displays information about the remote server testiq (output has been truncated):

```
sp_servercaps testiq
```

```
1,'Alter table with add','T'
2,'Alter table with drop','T'
3,'Owner supported','T'
4,'Primary key requires index','F'
5,'Create table constraints','T'
6,'Truncate table','T'
7,'Create index','T' 7,'Create index','T'
8,'Create unique index','T'
9,'Syscapability system table initialized','T'
10,'Subquery','T'
11,'Subquery in group by','T'
12,'Subquery in comparison','T'
13,'Subquery in exist','T'
14,'Subquery in IN','T'
15,'Subquery correlated','T'
16,'Subquery in select list','T'
17,'Subquery in update','T'
20,'Order by','T'
21,'Order by expressions','T'
22,'Order by column not in select list','T'
23,'Order by allowed in update','T'
25,'Joins','T'
26,'Outer joins','T'
27,'Full outer joins','T'
28,'Multiple outer joins','T'
29,'Logical operators in outer join','T'
30,'Outer joins mixed with normal joins','T'
31,'ANSI join syntax','T'
32,'TSQL join syntax','F'
33,'ODBC outer join syntax','F'
34,'Unrestricted ANSI ON','T'
40,'Group by','T'
41,'Group by ALL','T'
45,'Aggregates','T'
46,'Aggregates with column name','T'
50,'And','T'
51,'Or','T'
52,'Like','T'
53,'Like - TSQL','T'
54,'Distinct','T'
55,'In','T'
```

Standards and compatibility

Sybase Supported by Open Client/Open Server.

See also

CREATE SERVER statement in *Reference: Statements and Options*

Chapter 4, "Accessing Remote Data" and Chapter 5, "Server Classes for Remote Data Access" in the *System Administration Guide: Volume 2*

sp_tsql_environment system procedure

Function

To set connection options when users connect from jConnect or Open Client applications.

Syntax

sp_tsql_environment

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

If the connection uses the TDS communication protocol (that is, if it is an Open Client connection), sp_login_environment calls sp_tsql_environment.

This procedure sets database options so that they are compatible with default Sybase Adaptive Server Enterprise behavior.

To change the default behavior, create new procedures and alter your LOGIN_PROCEDURE option to point to these new procedures.

For more information about setting LOGIN_PROCEDURE to the name of a new procedure, see Chapter 3, "Sybase IQ as a Data Server" in the *System Administration Guide: Volume 2*.

Here is the text of sp_tsql_environment:

```
create procedure dbo.sp_tsql_environment()
begin
if db_property('IQStore') = 'Off' then

    -- ASA datastore
set temporary option close_on_endtrans = 'OFF'
end if;
set temporary option ansinull = 'OFF';
set temporary option tsql_variables = 'ON';
set temporary option ansi_blanks = 'ON';
set temporary option chained = 'OFF';
set temporary option quoted_identifier = 'OFF';
set temporary option allow_nulls_by_default = 'OFF';
set temporary option on_tsql_error = 'CONTINUE';
set temporary option isolation_level = '1';
set temporary option date_format = 'YYYY-MM-DD';
set temporary option timestamp_format = 'YYYY-MM-DD
HH:NN:SS.SSS';
set temporary option time_format = 'HH:NN:SS.SSS';
```

```
set temporary option date_order = 'MDY';
set temporary option escape_character = 'OFF'
end
```

See also

LOGIN_PROCEDURE option in Chapter 2, “Database Options,” in *Reference: Statements and Options*

Adaptive Server Enterprise system and catalog procedures

Adaptive Server Enterprise provides system and catalog procedures to carry out many administrative functions and to obtain system information. Sybase IQ has implemented support for some of these procedures.

System procedures are built-in stored procedures used for getting reports from and updating system tables. Catalog stored procedures retrieve information from the system tables in tabular form.

Note While these procedures perform the same functions as they do in Adaptive Server Enterprise and pre-version 12 Sybase IQ, they are not identical. If you have preexisting scripts that use these procedures, you might want to examine the procedures. To see the text of a stored procedure, run:

```
sp_helptext 'owner.procedure_name'
```

For all system stored procedures delivered by Sybase, the owner is dbo. To see the text of a stored procedure of the same name owned by a different user, you must specify that user, for example:

```
sp_helptext 'myname.myprocedure'
```

Adaptive Server Enterprise system procedures

Table 7-79 describes the Adaptive Server Enterprise system procedures provided in Sybase IQ.

Table 7-79: ASE system procedures provided in Sybase IQ

System procedure	Description
sp_addgroup *group-name*	Adds a group to a database
sp_addlogin *userid, password[, defdb [, deflanguage [, fullname]]]*	Adds a new user account to a database
sp_addmessage *message-num, message_text [, language]*	Adds user-defined messages to SYSUSERMESSAGES for use by stored procedure PRINT and RAISERROR calls
sp_addtype *typename, data-type, [, "identity" | nulltype]*	Creates a user-defined data type. Sybase IQ does not support IDENTITY columns.
sp_adduser *userid [, name_in_db [, grpname]]*	Adds a new user to a database
sp_changegroup *new-group-name, userid*	Changes a user's group or adds a user to a group
sp_dboption *[dbname, optname, {true | false}]*	Displays or changes database options
sp_dropgroup *group-name*	Drops a group from a database
sp_droplogin *userid*	Drops a user from a database
sp_dropmessage *message-number [, language]*	Drops user-defined messages
sp_droptype *typename*	Drops a user-defined data type
sp_dropuser *userid*	Drops a user from a database
sp_getmessage *message-num, @msg-var output [, language]*	Retrieves stored message strings from SYSUSERMESSAGES for PRINT and RAISERROR statements.
sp_helptext '*owner.object-name*'	Displays the text of a system procedure or view
sp_password *caller_passwd, new_passwd [, userid]*	Adds or changes a password for a user ID

Note Procedures like sp_dropuser provide minimal compatibility with Adaptive Server Enterprise stored procedures. If you are accustomed to Adaptive Server Enterprise (or Sybase IQ 11.x) stored procedures, compare their text with Sybase IQ 12 procedures before using the procedure in dbisql. To compare, use the command:

```
sp_helptext 'owner.procedure_name'
```

For system stored procedures delivered by Sybase, the owner is always dbo. To see the text of a stored procedure of the same name owned by a different user, you must specify that user, for example:

```
sp_helptext 'myname.myprocedure'
```

Adaptive Server Enterprise catalog procedures

Sybase IQ implements most of the Adaptive Server Enterprise catalog procedures with the exception of the sp_column_privileges procedure. The implemented catalog procedures are described in Table 7-80. Sybase IQ also has similar customized stored procedures for some of these Adaptive Server catalog procedures.

Table 7-80: ASE catalog procedures implemented in Sybase IQ

ASE catalog procedure	Description	IQ procedure
sp_columns *table-name [, table-owner] [, table-qualifier] [, column-name]*	Returns the data types of the specified column	
sp_fkeys *pktable_name [, pktable-owner][, pktable-qualifier] [, fktable-name] [, fktable_owner] [, fktable-qualifier]*	Returns foreign-key information about the specified table	
sp_pkeys *table-name [, table_owner] [, table_qualifier]*	Returns primary-key information for a single table	sp_iqpkeys
sp_special_columns *table_name [, table-owner] [, table-qualifier] [, col-type]*	Returns the optimal set of columns that uniquely identify a row in a table	
sp_sproc_columns *proc-name [, proc_owner] [, proc-qualifier] [, column-name]*	Returns information about the input and return parameters of a stored procedure	sp_iqprocparm
sp_stored_procedures *[sp-name] [, sp-owner] [, sp-qualifier]*	Returns information about one or more stored procedures	sp_iqprocedure
sp_tables *table-name [, table-owner] [, table-qualifier] [, table-type]*	Returns a list of objects that can appear in a FROM clause	

The following Adaptive Server Enterprise catalog procedures are not supported:

- sp_column_privileges
- sp_databases
- sp_datatype_info
- sp_server_info

SQL Anywhere supported procedures

Sybase IQ supports the SQL Anywhere procedures listed in “Alphabetical list of system procedures ” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > System procedures*.

Note In the section “System extended stored procedures” in *SQL Anywhere Server – SQL Reference*, the documentation states that users must be granted EXECUTE permission or have DBA authority. Several of the subsequent procedures, however, show permissions listed as none.

The procedure below requires DBA authority.

- xp_cmdshell – system extended procedure that allows a database server to execute external shell commands

CHAPTER 8

System Views

About this chapter

This chapter lists predefined views for Sybase IQ system tables.

About system views

The system tables are designed for internal use. To view the contents of the system tables, use the system views. A number of predefined system views are provided that present the information in the system tables in a readable format.

The definitions for the system views are included with their descriptions. Some of these definitions are complicated, but you do not need to understand them to use the views. They serve as good examples of what can be accomplished using the SELECT command and views.

About consolidated views

Consolidated views provide data in a form more frequently required by users. For example, consolidated views often provide commonly needed joins. Consolidated views differ from system views in that they are not just a straightforward view of raw data in an underlying system table. For example, many of the columns in the system views are unintelligible ID values, whereas in the consolidated views, they are readable names.

Consolidated views such as SYSCATALOG and SYSINDEXES are common to both Sybase IQ and SQL Anywhere. For definitions of these and other consolidated views, see "Consolidated views" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views*.

About compatibility views

Compatibility views are deprecated views provided for compatibility with earlier versions of SQL Anywhere and Sybase IQ. Where possible, use system views and consolidated views instead of compatibility views, as support for compatibility views may be eliminated in future versions of Sybase IQ.

For detailed information on compatibility views, see "Compatibility views" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views*.

About ASE T-SQL compatibility views

Sybase IQ provides a set of views owned by the special user DBO, which correspond to the Adaptive Server Enterprise system tables and views. See "Transact-SQL compatibility views" on page 587.

SYSARTICLE system view

Each row of the SYSARTICLE system view describes an article in a publication. The underlying system table for this view is ISYSARTICLE.

The SYSARTICLE view is a SQL Anywhere system view. See "SYSARTICLE system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSARTICLECOL system view

Each row of the SYSARTICLECOL system view identifies a column in an article. The underlying system table for this view is ISYSARTICLECOL.

The SYSARTICLECOL view is a SQL Anywhere system view. See "SYSARTICLECOL system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSARTICLECOLS consolidated view

Each row in the view identifies a column in an article.

The SYSARTICLECOLS view is a SQL Anywhere consolidated view. See "SYSARTICLECOLS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSARTICLES consolidated view

Each row in the SYSARTICLES view describes an article in a publication.

The SYSARTICLES view is a SQL Anywhere consolidated view. See "SYSARTICLES consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSCAPABILITIES consolidated view

Each row in the SYSCAPABILITIES view describes a capability. This view gets its data from the ISYSCAPABILITY and ISYSCAPABILITYNAME system tables.

The SYSCAPABILITIES view is a SQL Anywhere consolidated view. See "SYSCAPABILITIES consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSCAPABILITY system view

Each row of the SYSCAPABILITY system view identifies a capability of a remote server. The underlying system table for this view is ISYSCAPABILITY.

The SYSCAPABILITY view is a SQL Anywhere system view. See "SYSCAPABILITY system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSCAPABILITYNAME system view

Each row in the SYSCAPABILITYNAME system view names a capability that is defined in the SYSCAPABILITY system view.

The SYSCAPABILITYNAME view is a SQL Anywhere system view. See “SYSCAPABILITYNAME system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSCATALOG consolidated view

Each row in the SYSCATALOG view describes a system table.

The SYSCATALOG view is a SQL Anywhere consolidated view. See “SYSCATALOG consolidated view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSCHECK system view

Each row in the SYSCHECK system view provides the definition for a named check constraint in a table. The underlying system table for this view is ISYSCHECK.

The SYSCHECK view is a SQL Anywhere system view. See “SYSCHECK system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSCOLAUTH consolidated view

Each row in the SYSCOLAUTH view describes the set of privileges (UPDATE, SELECT, or REFERENCES) granted on a column. The SYSCOLAUTH view provides a user-friendly presentation of data in the SYSCOLPERM system view.

The SYSCOLAUTH view is a SQL Anywhere consolidated view. See “SYSCOLAUTH consolidated view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSCOLPERM system view

The GRANT statement can give UPDATE, SELECT, or REFERENCES permission to individual columns in a table. Each column with UPDATE, SELECT, or REFERENCES permission is recorded in one row of the SYSCOLPERM system view. The underlying system table for this view is ISYSCOLPERM.

The SYSCOLPERM view is a SQL Anywhere system view. See “SYSCOLPERM system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSCOLLATION compatibility view (deprecated)

The SYSCOLLATION compatibility view contains the collation sequence information for the database. It is obtainable via built-in functions and is not kept in the catalog.

The SYSCOLLATION view is a SQL Anywhere compatibility view. See “SYSCOLLATION compatibility view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Views > Compatibility views*.

SYSCOLLATIONMAPPINGS compatibility view (deprecated)

The SYSCOLLATIONMAPPINGS compatibility view contains only one row with the database collation mapping. It is obtainable via built-in functions and is not kept in the catalog.

The SYSCOLLATIONMAPPINGS view is a SQL Anywhere compatibility view. See "SYSCOLLATIONMAPPINGS compatibility view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views*.

SYSCOLSTAT system view

The SYSCOLSTAT system view contains the column statistics, including histograms, that are used by the optimizer. The contents of this view are best retrieved using the sa_get_histogram stored procedure or the Histogram utility. The underlying system table for this view is ISYSCOLSTAT.

The underlying system table for this view is always encrypted to protect the data from unauthorized access

The SYSCOLSTAT view is a SQL Anywhere system view. See "SYSCOLSTAT system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSCOLSTATS consolidated view

The SYSCOLSTATS view contains the column statistics that are stored as histograms and used by the optimizer.

The SYSCOLSTATS view is a SQL Anywhere consolidated view. See "SYSCOLSTATS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSCOLUMN compatibility view (deprecated)

The SYSCOLUMN view is provided for compatibility with older versions of Sybase IQ that offered a SYSCOLUMN system table. However, the previous SYSCOLUMN table has been replaced by the ISYSTABCOL system table, and its corresponding SYSTABCOL system view, which you should use instead.

The SYSCOLUMN view is a SQL Anywhere compatibility view. See "SYSCOLUMN compatibility view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views*.

SYSCOLUMNS consolidated view

Each row in the SYSCOLUMNS view describes one column of each table and view in the catalog.

The SYSCOLUMNS view is a SQL Anywhere consolidated view. See "SYSCOLUMNS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSCOLUMNS ASE compatibility view

This view is owned by user DBO. syscolumns contains one row for every column in every table and view, and a row for each parameter in a procedure. See Table 8-1 on page 588.

SYSCOMMENTS ASE compatibility view

This view is owned by user DBO. syscomments contains entries for each view, rule, default, trigger, table constraint, partition, procedure, computed column, function-based index key, and other forms of compiled objects. The text column contains the original definition statements. If the text column is longer than 255 bytes, the entries span rows. Each object can occupy as many as 65,025 rows. See Table 8-1 on page 588.

SYSCONSTRAINT system view

Each row in the SYSCONSTRAINT system view describes a named constraint in the database. The underlying system table for this view is ISYSCONSTRAINT.

The SYSCONSTRAINT view is a SQL Anywhere system view. See "SYSCONSTRAINT system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSDBFILE system view

Each row in the SYSDBFILE system view describes a dbspace file. The underlying system table for this view is ISYSDBFILE.

Note This view replaces the deprecated system view SYSFILE.

The SYSDBFILE view is a SQL Anywhere system view. See "SYSDBFILE system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSDBSPACE system view

Each row in the SYSDBSPACE system view describes a dbspace file. The underlying system table for this view is ISYSDBSPACE.

Note This view replaces the deprecated system view SYSFILE.

The SYSDBSPACE view is a SQL Anywhere system view. See "SYSDBSPACE system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSDBSPACEPERM system view

Each row in the SYSDBSPACEPERM system view describes a permission on a dbspace file. The underlying system table for this view is ISYSDBSPACEPERM.

The SYSDBSPACEPERM view is a SQL Anywhere system view. See "SYSDBFILE system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSDEPENDENCY system view

Each row in the SYSDEPENDENCY system view describes a dependency between two database objects. The underlying system table for this view is ISYSDEPENDENCY.

A dependency exists between two database objects when one object references another object in its definition. For example, if the query specification for a view references a table, the view is said to be dependent on the table. The database server tracks dependencies of views on tables, views, materialized views, and columns

The SYSDEPENDENCY view is a SQL Anywhere system view. See "SYSDEPENDENCY system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSDOMAIN system view

The SYSDOMAIN system view records information about built-in data types (also called domains). The contents of this view does not change during normal operation. The underlying system table for this view is ISYSDOMAIN.

The SYSDOMAIN view is a SQL Anywhere system view. See "SYSDOMAIN system view" in theSQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSEVENT system view

Each row in the SYSEVENT system view describes an event created with CREATE EVENT. The underlying system table for this view is ISYSEVENT.

The SYSEVENT view is a SQL Anywhere system view. See "SYSEVENT system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSEVENTTYPE system view

The SYSEVENTTYPE system view defines the system event types that can be referenced by CREATE EVENT.

The SYSEVENTTYPE view is a SQL Anywhere system view. See "SYSEVENT system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSEXTERNENV system view

Sybase IQ includes support for six external runtime environments. These include embedded SQL and ODBC applications written in C/C++, and applications written in Java, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

Each row in the SYSEXTERNENV system view describes the information needed to identify and launch each of the external environments. The underlying system table for this view is ISYSEXTERNENV.

The SYSEXTERNENV view is a SQL Anywhere system view. See “SYSEXTERNENV system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSEXTERNENVOBJECT system view

Sybase IQ includes support for six external runtime environments. These include embedded SQL and ODBC applications written in C/C++, and applications written in Java, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

Each row in the SYSEXTERNENVOBJECT system view describes an installed external object. The underlying system table for this view is ISYSEXTERNENVOBJECT.

The SYSEXTERNENVOBJECT view is a SQL Anywhere system view. See “SYSEXTERNENVOBJECT system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSEXTERNLOGIN system view

Each row in the SYSEXTERNLOGIN system view describes an external login for remote data access. The underlying system table for this view is ISYSEXTERNLOGIN.

The SYSEXTERNLOGIN view is a SQL Anywhere system view. See "SYSEXTERNLOGIN system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSFILE compatibility view (deprecated)

Each row in the SYSFILE system view describes a dbspace for a database. Every database consists of one or more dbspaces; each dbspace corresponds to an operating system file.

The SYSFILE view is a SQL Anywhere compatibility view. See "SYSFILE compatibility view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSFKCOL compatibility view (deprecated)

Each row of SYSFKCOL describes the association between a foreign column in the foreign table of a relationship and the primary column in the primary table. This view is deprecated; use the SYSIDX and SYSIDXCOL system views instead.

The SYSFKCOL view is a SQL Anywhere compatibility view. See "SYSFKCOL compatibility view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSFKEY system view

Each row in the SYSFKEY system view describes a foreign key constraint in the system. The underlying system table for this view is ISYSFKEY.

The SYSFKEY view is a SQL Anywhere system view. See “SYSFKEY system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSFOREIGNKEY compatibility view (deprecated)

The SYSFOREIGNKEY view is provided for compatibility with older versions of Sybase IQ that offered a SYSFOREIGNKEY system table. However, the previous SYSFOREIGNKEY system table has been replaced by the ISYSFKEY system table, and its corresponding SYSFKEY system view, which you should use instead.

The SYSFOREIGNKEY view is a SQL Anywhere consolidated view. See “SYSFOREIGNKEY consolidated view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSFOREIGNKEYS consolidated view

Each row in the SYSFOREIGNKEYS view describes one foreign key for each table in the catalog.

The SYSFOREIGNKEYS view is a SQL Anywhere consolidated view. See “SYSFOREIGNKEYS consolidated view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSGROUP system view

There is one row in the SYSGROUP system view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups. The underlying system table for this view is ISYSGROUP.

The SYSGROUP view is a SQL Anywhere system view. See "SYSGROUP system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSGROUPS consolidated view

There is one row in the SYSGROUPS view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups.

The SYSGROUPS view is a SQL Anywhere consolidated view. See "SYSGROUPS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSHISTORY system view

Each row in the SYSHISTORY system view records a system operation on the database, such as a database start, a database calibration, and so on. The underlying system table for this view is ISYSHISTORY.

The SYSHISTORY view is a SQL Anywhere system view. See "SYSHISTORY system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSIDX system view

Each row in the SYSIDX system view defines a logical index in the database. The underlying system table for this view is ISYSIDX.

Note This view replaces the deprecated system view SYSINSDEX.

The SYSIDX view is a SQL Anywhere system view. See "SYSIDX system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSIDXCOL system view

Each row in the SYSIDXCOL system view describes one column of an index described in the SYSIDX system view. The underlying system table for this view is ISYSIDXCOL.

The SYSIDXCOL view is a SQL Anywhere system view. See "SYSIDXCOL system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSINDEX compatibility view (deprecated)

The SYSINDEX view is provided for compatibility with older versions of Sybase IQ that offered a SYSINDEX system table. However, the SYSINDEX system table has been replaced by the ISYSIDX system table, and its corresponding SYSIDX system view, which you should use instead.

The SYSINDEX view is a SQL Anywhere compatibility view. See "SYSINDEX compatibility view (deprecated)" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views*.

SYSINDEXES consolidated view

Each row in the SYSINDEXES view describes one index in the database. As an alternative to this view, you could also use the SYSIDX and SYSIDXCOL system views.

The SYSINDEXES view is a SQL Anywhere consolidated view. See "SYSINDEXES consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSINDEXES ASE compatibility view

This view is owned by user DBO. sysindexes contains one row for each clustered index, one row for each nonclustered index, one row for each table that has no clustered index, and one row for each table that contains text or image columns.This table also contains one row for each function-based index or index created on a computed column. See Table 8-1 on page 588.

SYSINFO compatibility view (deprecated)

The SYSINFO view indicates the database characteristics, as defined when the database was created. It always contains only one row. This view is obtainable via built-in functions and is not kept in the catalog.

The SYSINFO view is a SQL Anywhere compatibility view. See "SYSINFO compatibility view (deprecated)" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views*.

SYSIQBACKUPHISTORY system view

This view presents group information from ISYSIQBACKUPHISTORY in a readable format. Each row in this view describes a particular backup operation that finished successfully.

The view SYSIQBACKUP projects equivalent string values for columns type, subtype, and bkp_virtual.

Column name	Column type	Column constraint	Description
bu_id	unsigned bigint	NOT NULL	Transaction identifier of the checkpoint of the operation. Backup ID for backup operations.
bu_time	timestamp	NOT NULL	Time of backup operation that is recorded in backup record.
type	tinyint	NOT NULL	Backup type: 0 = FULL 1 = INCREMENTAL 2 = INCREMENTAL SINCE FULL
selective_type	tinyint	NOT NULL	Backup subtype: 0 = ALL (backs up all dbfiles) 1 = READ/WRITE ONLY (backs up all read-write files) 2 = READ ONLY (backs up a particular read-only file)
virtual_type	tinyint	NOT NULL	Backup virtual type: 0 = NONE 1 = DECOUPLED 2 = ENCAPSULATED
dependson_id	unsigned bigint	NULL	NULL for FULL backup
cmd	long varchar	NOT NULL	Full text of command
creator	char(128)	NOT NULL	User who issued backup command
version	unsigned int	NOT NULL	Backup version

Constraints on underlying system table

Primary key (bu_id)

SYSIQBACKUPHISTORYDETAIL system view

This view describes all the dbfile records present in the database at backup time. Each row in this view describes a particular backup operation that finished successfully. It presents group information from ISYSIQBACKUPHISTORYDETAIL in a readable format. The column constraint for each column is NOT NULL.

Column name	Column type	Description
bu_id	unsigned bigint	Transaction identifier of the checkpoint of the operation. Backup ID for backup operation
dbspace_id	smallint	The dbspace ID of which this dbfile record is associated.
dbfile_id	smallint	The dbfile ID present in dbspace during ongoing backup operation
dbspace_rwstatus	char(1)	T indicates read-write
dbspace_createid	unsigned bigint	The transaction ID of the transaction that created the dbspace
dbspace_alterid	unsigned bigint	Transaction ID that marked the dbspace RO. If not marked, then the create ID
dbspace_online	char(1)	T indicates online
dbfile_rwstatus	char(1)	T indicates read-write
dbfile_createid	unsigned bigint	The transaction ID of the transaction that created this dbfile
dbfile_alterid	unsigned bigint	The transaction ID of the transaction that last altered the read-write status of this dbfile
is_backed_up	char(1)	Indicates that the dbfile is backed up in this backup
start_block	unsigned bigint	Start block for the dbfile
num_blocks	unsigned bigint	Total number of blocks in dbfile
num_blocks_backed_up	unsigned bigint	Total number of blocks backed up

Column name	Column type	Description
dbspace_name	char(128)	Dbspace name
dbfile_name	char(128)	Logical file name of the dbfile
dbfile_path	long varchar	Physical path of the file

Constraints on underlying system table

Primary key (bu_id, dbfile_id)

Foreign key (txn_id) references SYS.ISYSBACKUPHISTORY

SYSIQCOLUMN system view (deprecated)

SYSIQCOLUMN has been replaced by the SYSIQTABCOL system view. See “SYSIQTABCOL system view” on page 559.

SYSIQDBFILE system view

Presents group information from ISYSIQDBFILE in a readable format.

Note This view replaces the deprecated system view SYSIQFILE.

Column name	Column type	Description
dbfile_id	small int	Unique ID for the dbfile
start_block	rowid	Number of the first block
block_count	rowid	Number of blocks for this file (dbspace)
reserve_size	rowid	Preallocated file system space for the dbspace
allocated	char(1)	Defines whether the segment is preallocated (T) or autoallocated (F)

Column name	Column type	Description
data_offset	unsigned int	Identifies the byte location of where the Sybase IQ data starts, relative to the beginning of the raw partition
create_time	timestamp	Date and time the file was created
last_modified	timestamp	Date and time the file was last modified
read_write	char(1)	T indicates read-write
online	char(1)	T indicates online
create_txn_id	xact_id	Transaction ID that created the dbfile
alter_txn_id	xact_id	Transaction ID that last modified read_write status
server_id	unsigned int	Multiplex server name
file_name	text	the IQ dbspace name used by the multiplex server to open the IQ dbspace

Constraints on underlying system table

Foreign key (server_id) references SYS.ISYSIQMPXSERVER

Unique (server_id, file_name)

SYSIQDBSPACE system view

Presents group information from ISYSIQDBSPACE in a readable format.

Column name	Column type	Description
dbspace_id	small int	Each dbspace in a database is assigned a unique number (dbspace ID)
last_modified	timestamp	Time at which the dbspace's read-write status was last modified
segment_type	char(8)	Segment type: Main, Temp or Msg

Column name	Column type	Description
read_write	char(1)	'T' – read writable, 'F' – read only
online	char(1)	'T' – online; 'F' – offline
create_txn_id	xact_id	Transaction ID that create the dbspace
alter_txn_id	xact_id	Transaction ID that last modified read_write status
striping_on	char(1)	'T' – disk striping on; 'F' – disk striping off
stripe_size_kb	unsigned int	Number of kilobytes written to each file of the dbspace before the disk striping algorithm moves to the next dbfile

Constraints on underlying system table

Primary key (dbspace_id)

Foreign key (dbspace_id) references SYS.ISYSDBSPACE(dbspace_id)

SYSIQFILE system view (deprecated)

SYSIQFILE has been replaced by the SYSIQDBFILE system view. See “SYSIQDBFILE system view” on page 549.

SYSIQIDX system view

Presents group information from ISYSIQIDX in a readable format. Each row in the SYSIQIDX view describes an IQ index.

Note This view replaces the deprecated system view SYSIQINDEX.

Column name	Column type	Description
table_id	unsigned int	The table number uniquely identifies the table to which this index applies
index_id	unsigned int	Each index for one particular table is assigned a unique index number
index_type	char(4)	Index type
index_owner	char(4)	Index owner
max_key	unsigned int	For internal use
identity_location	hs_vdorecid	For internal use
identity_size	unsigned int	For internal use
identity_location_size	unsigned int	For internal use
link_table_id	unsigned int	For internal use
link_index_id	unsigned int	For internal use
delimited_by	varchar(1024)	(WD indexes only) List of separators used to parse a column's string into the words to be stored in that column's WD index
limit	unsigned int	(WD indexes only) Maximum word length for WD index

Constraints on underlying system table

Primary key (table_id, index_id)

Foreign key (table_id, index_id) references SYS.ISYIDX

Foreign key (link_table_id, link_index_id, table_id, index_id) references SYS.ISYSIDX

SYSIQINFO system view

Presents group information from ISYSIQINFO in a readable format.

The ISYSIQINFO system table indicates the database characteristics as defined when the Sybase IQ database was created using CREATE DATABASE. It always contains only one row. The multiplex_name and last_multiplex_mode columns were added for Sybase IQ 15.0.

Column name	Column type	Description
create_time	TIMESTAMP NOT NULL	Date and time that the database was created.
update_time	TIMESTAMP NOT NULL	Date and time of the last update.
file_format_version	UNSIGNED INT NOT NULL	File format number of files for this database.
cat_format_version	UNSIGNED INT NOT NULL	Catalog format number for this database.
sp_format_version	UNSIGNED INT NOT NULL	Stored procedure format number for this database.
block_size	UNSIGNED INT NOT NULL	Block size specified for the database.
chunk_size	UNSIGNED INT NOT NULL	Number of blocks per page as determined by the block size and page size specified for the database.
file_format_date	CHAR(10) NOT NULL	Date when file format number was last changed.
dbsig	BINARY(136) NOT NULL	Used internally by catalog.
commit_txn_id	xact_id	For internal use.
rd_commit_txn_id	xact_id	For internal use.
multiplex name	CHAR(128) NULL	Name of the multiplex that this database is a member of.
last_multiplex_mode	TINYINT NULL	(Column unused in Sybase IQ 15.1) Mode of the server that last opened the catalog read-write. One of the following values. • 0 – Single Node. • 1 – Reader. • 2 – Coordinator. • 3 – Writer.

SYSIQJOINIDX system view

Presents group information from ISYSIQJOINIDX in a readable format. Each row in the SYSIQJOINIDX view describes an IQ join index.

Note This view replaces the deprecated system view SYSIQJOININDEX.

Column name	Column type	Description
joinindex_id	unsigned int	Each join index is assigned a unique number that is the primary key.
jvt_id	unsigned int	For internal use.
dbspace_id	smallint	ID of the dbspace.
joinindex_name	char(128)	Defines the name of the join index.
joinindex_type	char(12)	For internal use.
creator	unsigned int	The number of the user that created the join index.
join_info_location	hs_vdorecid	For internal use.
join_info_loc_size	unsigned int	For internal use.
join_info_size	unsigned int	For internal use.
block_map	hs_blockmapidentity	For internal use.
block_map_size	unsigned int	For internal use.
vdo	hs_vdoidentity	For internal use.
vdo_size	unsigned_int	For internal use.
commit_txn_id	xact_id	For internal use.
txn_id	xact_id	For internal use.
valid	char(1)	Indicates whether this join index needs to be synchronized. Y indicates that it does not require synchronization, N indicates that it does require synchronization.

Constraints on underlying system table

Primary key (joinindex_id)

Foreign key (jvt_id) references SYS.ISYSTAB

Foreign key (dbspace_id) references SYS.ISYSDBSPACE

Foreign key (creator) references SYS.ISYSUSER

Unique (jvt_id, commit_txn_id, txn_id)

SYSIQJOININDEX system view (deprecated)

SYSIQJOININDEX has been replaced by the SYSIQJOINIDX system view. See “SYSIQJOINIDX system view” on page 554.

SYSIQJOINIXCOLUMN system view

```
ALTER VIEW "SYS"."SYSIQJOINIXCOLUMN"
as select * from SYS.ISYSIQJOINIXCOLUMN
```

Presents group information from ISYSIQJOINIXCOLUMN in a readable format. Each row in the SYSIQJOINIXCOLUMN view describes an IQ join index.

Column name	Column type	Description
joinindex_id	unsigned bigint	Corresponds to a join index value in SYSIQJOINIDX.
left_table_id	unsigned int	Corresponds to a table value in SYSTAB that forms the left side of the join operation.
left_column_id	unsigned int	Corresponds to a column value in SYSTABCOL that is part of the left side of the join.
join_type	char(4)	Only value currently supported is “=”.
right_table_id	unsigned int	Corresponds to a table value in SYSTAB that forms the right side of the join operation
right_column_id	unsigned int	Corresponds to a column value in SYSTABCOL that is part of the right side of the join.

Column name	Column type	Description
order_num	unsigned int	For internal use.
left_order_num	unsigned int	For internal use.
right_order_num	unsigned int	For internal use.
key_type	char(8)	Defines the type of join on the keys. 'NATURAL' is a natural join, 'KEY' is a key join, 'ON' is a left outer/right outer/full join.
coalesce	char(1)	Not used.

Constraints on underlying system table

Primary key (joinindex_id, left_table_id, left_column_id, right_table_id, right_column_id)

Foreign key (joinindex_id) references SYS.ISYSIQJOINIDX

Foreign key (left_table_id, column_id) references SYS.ISYSTABCOL

Foreign key (right_table_id, column_id) references SYS.ISYSTABCOL

SYSIQJOINIXTABLE system view

```
ALTER VIEW "SYS"."SYSIQJOINIXTABLE"
as select * from SYS.ISYSIQJOINIXTABLE
```

Presents group information from ISYSIQJOINIXTABLE in a readable format. Each row in the SYSIQJOINIXTABLE view describes an IQ join index.

Column name	Column type	Description
table_id	unsigned int	Corresponds to a table value in SYSTAB that is included in a join operation.
joinindex_id	unsigned bigint	Corresponds to a join index value in SYSIQJOINIDX.
active	unsigned int	Defines the number of times the table is used in the join index.

Constraints on underlying system table

Primary key (table_id, joinindex_id)

Foreign key (table_id) references SYS.ISYSTAB

Foreign key (joinindex_id) references SYS.ISYSIQJOINIDX

SYSIQMPXLOGINPOLICYOPTION system view

See “SYSIQMPXLOGINPOLICYOPTION system view” in Appendix A, “Multiplex Reference,” in *Using Sybase IQ Multiplex*.

SYSIQMPXSERVER system view

See “SYSIQMPXSERVER system view,” in Appendix A, “Multiplex Reference,” in *Using Sybase IQ Multiplex*.

SYSIQOBJECTS ASE compatibility view

SYSIQOBJECTS presents one row for each system table, user table, view, procedure, trigger, event, join index, constraint, domain (sysdomain), domain (sysusertype), column, and index. This view is owned by user DBO. See Table 8-1 on page 588.

SYSIQPARTITIONCOLUMN system view

```
ALTER VIEW "SYS"."SYSIQPARTITIONCOLUMN"
as select * from SYS.ISYSIQPARTITIONCOLUMN
```

Presents group information from ISYSIQPARTITIONCOLUMN in a readable format. Each row in the SYSIQPARTITIONCOLUMN view describes a column in a partition described in the SYSIQPARTITION view in a partitioned table described in the SYSPARTITIONSCHEME view. SYSIQPARTITIONCOLUMN only describes partitions of columns that are not stored on the dbspace of the partition.

Column name	Column type	Description
partitioned_object_id	unsigned bigint	Unique ID assigned to each partitioned object (table)
partition_id	unsigned int	Identifies a partition in a partitioned table.
column_id	unsigned int	The column ID of the column.
dbspace_id	smallint	The dbspace ID of the dbspace where this column of the partition is stored.

Constraints on underlying system table

Primary key (partitioned_object_id, partition_id, column_id)

Foreign key (partitioned_object_id, partition_id) references SYS.ISYSPARTITION

Foreign key (dbspace_id) references SYS.ISYSDBSPACE

SYSIQTAB system view

```
ALTER VIEW "SYS"."SYSIQTAB"
as select * from SYS.ISYSIQTAB
```

Presents group information from ISYSIQTAB in a readable format. Each row in the SYSIQTAB view describes an IQ table.

Note This view replaces the deprecated system view SYSIQTABLE.

Column name	Column type	Description
table_id	unsigned int	Each table is assigned a unique number (the table number) that is the primary key.
block_map	hs_blockmapidentity	For internal use.

Column name	Column type	Description
block_map_size	unsigned int	For internal use.
vdo	hs_vdoidentity	For internal use.
vdoid_size	unsigned int	For internal use.
info_location	hs_vdorecid	Not used. Always zero.
info_recid_size	unsigned int	Not used. Always zero.
info_location_size	unsigned int	Not used. Always zero.
commit_txn_id	xact_id	For internal use.
txn_id	xact_id	For internal use.
join_id	unsigned int	For internal use.
update_time	timestamp	Last date and time the IQ table was modified.

Constraints on underlying system table

Primary key (table_id)

SYSIQTABCOL system view

```
ALTER VIEW "SYS"."SYSIQTABCOL"
as select * from SYS.ISYSIQTABCOL
```

Presents group information from ISYSIQTABCOL in a readable format. Each row in the SYSIQTABCOL view describes a column in an IQ table.

Note This view replaces the deprecated system view SYSIQCOLUMN.

Column name	Column type	Description
table_id	unsigned int	The table number uniquely identifies the table to which this column belongs. It corresponds to the table_id column of SYSTAB.

Column name	Column type	Description
column_id	unsigned int	Each table starts numbering columns at 1. The order of column numbers determines the order that columns are displayed in the command **select * from table.**
link_table_id	unsigned int	For internal use.
link_column_id	unsigned int	For internal use.
max_length	unsigned int	Indicates the maximum length allowed by the column.
approx_unique_count	rowid	Approximate number of unique values (cardinality) of this column.
cardinality	rowid	The actual number of unique values (cardinality) of this column.
has_data	char(1)	Indicates that the column contains data (T/F).
has_original	char(1)	Indicates the join index has the original data (T/F).
original_not_null	char(1)	Indicates the join index column with the original data was NOT NULL (T/F).
original_unique	char(1)	Indicates the join index column with the original data was UNIQUE (T/F).

Constraints on underlying system table

Primary key (table_id)

SYSIQTABLE system view (deprecated)

SYSIQTABLE has been replaced by the SYSIQTAB system view. See “SYSIQTAB system view” on page 558.

SYSIQVINDEX ASE compatibility view

SYSIQVINDEX provides one row for each non-FP IQ index. This view is owned by user DBO. See Table 8-1 on page 588.

SYSIXCOL compatibility view (deprecated)

The SYSIXCOL view is provided for compatibility with older versions of Sybase IQ that offered a SYSIXCOL system table. However, the SYSIXCOL system table has been replaced by the ISYSIDXCOL system table, and its corresponding SYSIDXCOL system view.

The SYSIXCOL view is a SQL Anywhere compatibility view. See “SYSIXCOL compatibility view (deprecated)” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views*.

SYSJAR system view

Each row in the SYSJAR system view defines a JAR file stored in the database. The underlying system table for this view is ISYSJAR.

The SYSJAR view is a SQL Anywhere system view. See “SYSJAR system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSJARCOMPONENT system view

Each row in the SYSJARCOMPONENT system view defines a JAR file component. The underlying system table for this view is ISYSJARCOMPONENT.

The SYSJARCOMPONENT view is a SQL Anywhere system view. See "ISYSJARCOMPONENT system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSJAVACLASS system view

Each row in the SYSJAVACLASS system view describes one Java class stored in the database. The underlying system table for this view is ISYSJAVACLASS.

The SYSJAVACLASS view is a SQL Anywhere system view. See "SYSJAVACLASS system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSLOGINMAP system view

The SYSLOGINMAP system view contains one row for each user that can connect to the database using either an integrated login, or a Kerberos login. As a security measure, only users with DBA authority can view the contents of this view. The underlying system table for this view is ISYSLOGINMAP.

The SYSLOGINMAP view is a SQL Anywhere system view. See "SYSLOGINMAP system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSLOGINPOLICY system view

The underlying system table for this view is ISYSLOGINPOLICY.

The SYSLOGINPOLICY view is a SQL Anywhere system view. See "SYSLOGINPOLICY system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSLOGINPOLICYOPTION system view

The underlying system table for this view is ISYSLOGINPOLICYOPTION.

The SYSLOGINPOLICYOPTION view is a SQL Anywhere system view. See "SYSLOGINPOLICYOPTION system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSLOGINS ASE compatibility view

This view is owned by user DBO. syslogins contains one row for each valid Adaptive Server user account. See Table 8-2 on page 590.

SYSMVOPTION system view

Each row in the SYSMVOPTION system view describes the setting of one option value for a materialized view at the time of its creation. However, the description does not contain the option name. The underlying system table for this view is ISYSMVOPTION.

The SYSMVOPTION view is a SQL Anywhere system view. See "SYSMVOPTION system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSMVOPTIONNAME system view

Each row in the SYSMVOPTIONNAME system view contains the name of an option defined in the SYSMVOPTION system view. The underlying system table for this view is ISYSMVOPTIONNAME.

The SYSMVOPTIONNAME view is a SQL Anywhere system view. See “SYSMVOPTIONNAME system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSOBJECT system view

Each row in the SYSOBJECT system view describes a database object. The underlying system table for this view is ISYSOBJECT.

The SYSOBJECT view is a SQL Anywhere system view. See “SYSOBJECT system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSOBJECTS ASE compatibility view

This view is owned by user DBO. sysobjects contains one row for each table, view, stored procedure, extended stored procedure, log, rule, default, trigger, check constraint, referential constraint, computed column, function-based index key, and temporary object, and other forms of compiled objects. It also contains one row for each partition condition ID when object type is N. See Table 8-1 on page 588.

SYSOPTION system view

The SYSOPTION system view contains one row for each option setting stored in the database. Each user can have their own setting for a given option. In addition, settings for the PUBLIC user ID define the default settings to be used for users that do not have their own setting. The underlying system table for this view is ISYSOPTION.

The SYSOPTION view is a SQL Anywhere system view. See "SYSOPTION system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSOPTIONS consolidated view

Each row in the SYSOPTIONS view describes one option that was created using the SET command. Each user can have their own setting for each option. In addition, settings for the PUBLIC user define the default settings to be used for users that do not have their own setting.

The SYSOPTIONS view is a SQL Anywhere consolidated view. See "SYSOPTIONS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSOPTSTAT system view

The SYSOPTSTAT system view stores the cost model calibration information as computed by the ALTER DATABASE CALIBRATE statement. The contents of this view are for internal use only and are best accessed via the sa_get_dtt system procedure. The underlying system table for this view is ISYSOPTSTAT.

The SYSOPTSTAT view is a SQL Anywhere system view. See "SYSOPTSTAT system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSPARTITION system view

```
ALTER VIEW "SYS"."SYSPARTITION"
as select * from SYS.ISYSPARTITION
```

Presents group information from ISYSPARTITION in a readable format. Each row in the SYSPARTITION view describes a partitioned object (table or index) in the database.

Column name	Column type	Description
partitioned_object_id	unsigned bigint	Unique number assigned to each partitioned object (table)
partition_id	unsigned int	Identifies a partition in a partitioned table.
partition_object_id	unsigned bigint	Each table partition is an object itself and is assigned a unique number from the table object or index object.
partition_values	long varchar	Contains the upper bound for this range partition.
position	unsigned int	Ordinal number of partition.
partition_name	char(128)	Name of partition

Constraints on underlying system table

Primary key (partitioned_object_id, partition_id)

Unique (partition_object_id, position)

Foreign key (partition_object_id) references SYS.ISYSOBJECT

Foreign key (partitioned_object_id) references SYS.ISYSOBJECT

SYSPARTITIONKEY system view

```
ALTER VIEW "SYS"."SYSPARTITIONKEY"
as select * from SYS.ISYSPARTITIONKEY
```

Presents group information from ISYSPARTITIONKEY in a readable format. Each row in the SYSPARTITIONKEY view describes a partitioned object (table or index) in the database.

Column name	Column type	Description
partitioned_object_id	unsigned bigint	Each partitioned object (table) is assigned a unique object number.

Column name	Column type	Description
column_id	unsigned int	The column ID identifies the table column as part of the partitioning key.
position	smallint	Position of this column in the partitioning key. Position is 0 based. A position of 0 indicates the 1st column in the partitioning key.

Constraints on underlying system table

Primary key (partitioned_object_id, column_id)

Foreign key (partitioned_object_id) references SYS.ISYSOBJECT

SYSPARTITIONSCHEME system view

```
ALTER VIEW "SYS"."SYSPARTITIONSCHEME"
as select * from SYS.ISYSPARTITIONSCHEME
```

Presents group information from ISYSPARTITIONSCHEME in a readable format. Each row in the SYSPARTITIONSCHEME view describes a partitioned object (table or index) in the database.

Column name	Column type	Description
partitioned_object_id	unsigned bigint	Each partitioned object (table) is assigned a unique number
partition_method	tinyint	The partitioning method for this table. Valid values: 1– for range Sybase IQ supports only range partitioning.
subpartition_method	tinyint	Reserved for future use.

Constraints on underlying system table

Primary key (partitioned_object_id)

Foreign key (partitioned_object_id) references SYS.ISYSOBJECT

SYSPHYSIDX system view

Each row in the SYSPHYSIDX system view defines a physical index in the database. The underlying system table for this view is ISYSPHYSIDX.

The SYSPHYSIDX view is a SQL Anywhere system view. See "SYSPHYSIDX system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSPROCAUTH consolidated view

Each row in the SYSPROCAUTH view describes a set of privileges granted on a procedure. As an alternative, you can also use the SYSPROCPERM system view.

The SYSPROCAUTH view is a SQL Anywhere consolidated view. See "SYSPROCAUTH consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSPROCEDURE system view

Each row in the SYSPROCEDURE system view describes one procedure in the database. The underlying system table for this view is ISYSPROCEDURE.

The SYSPROCEDURE view is a SQL Anywhere system view. See "SYSPROCEDURE system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSPROCPARM system view

Each row in the SYSPROCPARM system view describes one parameter to a procedure in the database. The underlying system table for this view is ISYSPROCPARM.

The SYSPROCPARM view is a SQL Anywhere system view. See "SYSPROCPARM system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSPROCPARMS consolidated view

Each row in the SYSPROCPARMS view describes a parameter to a procedure in the database.

The SYSPROCPARMS view is a SQL Anywhere consolidated view. See "SYSPROCPARMS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSPROCPERM system view

Each row of the SYSPROCPERM system view describes a user who is granted permission to execute a procedure. Only users who have been granted permission can execute a procedure. The underlying system table for this view is ISYSPROCPERM.

The SYSPROCPERM view is a SQL Anywhere system view. See "SYSPROCPERM system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSPROCS consolidated view

The SYSPROCS view shows the procedure or function name, the name of its creator and any comments recorded for the procedure or function.

The SYSPROCS view is a SQL Anywhere consolidated view. See "SYSPROCS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSPROXYTAB system view

Each row of the SYSPROXYTAB system view describes the remote parameters of one proxy table. The underlying system table for this view is ISYSPROXYTAB.

The SYSPROXYTAB view is a SQL Anywhere system view. See "SYSPROXYTAB system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSPUBLICATION system view

Each row in the SYSPUBLICATION system view describes a SQL Remote or MobiLink publication. The underlying system table for this view is ISYSPUBLICATION.

The SYSPUBLICATION view is a SQL Anywhere system view. See "SYSPUBLICATION system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSPUBLICATIONS consolidated view

Each row in the SYSPUBLICATIONS view describes a SQL Remote or MobiLink publication.

The SYSPUBLICATIONS view is a SQL Anywhere consolidated view. See "SYSPUBLICATIONS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSREMARK system view

Each row in the SYSREMARK system view describes a remark (or comment) for an object. The underlying system table for this view is ISYSREMARK.

The SYSREMARK view is a SQL Anywhere system view. See "SYSREMARK system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSREMOTEOPTION system view

Each row in the SYSREMOTEOPTION system view describes the value of a SQL Remote message link parameter. The underlying system table for this view is ISYSREMOTEOPTION.

Some columns in this view contain potentially sensitive data. For that reason, access to this view is restricted to users with DBA authority. The SYSREMOTEOPTION2 view provides public access to the data in this view except for the potentially sensitive columns.

The SYSREMOTEOPTION view is a SQL Anywhere system view. See "SYSREMOTEOPTION system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSREMOTEOPTION2 consolidated view

Presents, in a readable format, the columns from SYSREMOTEOPTION and SYSREMOTEOPTIONTYPE that do not contain sensitive data.

The SYSREMOTEOPTION2 view is a SQL Anywhere consolidated view. See "SYSREMOTEOPTION2 consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSREMOTEOPTIONS consolidated view

Each row of the SYSREMOTEOPTIONS view describes the values of a SQL Remote message link parameter. Some columns in this view contain potentially sensitive data. For that reason, access to this view is restricted to users with DBA authority. The SYSREMOTEOPTION2 view provides public access to the insensitive data.

The SYSREMOTEOPTIONS view is a SQL Anywhere consolidated view. See "SYSREMOTEOPTIONS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSREMOTEOPTIONTYPE system view

Each row in the SYSREMOTEOPTIONTYPE system view describes one of the SQL Remote message link parameters. The underlying system table for this view is ISYSREMOTEOPTIONTYPE.

The SYSREMOTEOPTIONTYPE view is a SQL Anywhere system view. See "SYSREMOTEOPTIONTYPE system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSREMOTETYPE system view

The SYSREMOTETYPE system view contains information about SQL Remote. The underlying system table for this view is ISYSREMOTETYPE.

The SYSREMOTETYPE view is a SQL Anywhere system view. See “SYSREMOTETYPE system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSREMOTETYPES consolidated view

Each row of the SYSREMOTETYPES view describes one of the SQL Remote message types, including the publisher address.

The SYSREMOTETYPES view is a SQL Anywhere consolidated view. See “SYSREMOTETYPES consolidated view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSREMOTEUSER system view

Each row in the SYSREMOTEUSER system view describes a user ID with REMOTE permissions (a subscriber), together with the status of SQL Remote messages that were sent to and from that user. The underlying system table for this view is ISYSREMOTEUSER.

The SYSREMOTEUSER view is a SQL Anywhere system view. See “SYSREMOTEUSER system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSREMOTEUSERS consolidated view

Each row of the SYSREMOTEUSERS view describes a user ID with REMOTE permissions (a subscriber), together with the status of SQL Remote messages that were sent to and from that user.

The SYSREMOTEUSERS view is a SQL Anywhere consolidated view. See "SYSREMOTEUSERS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSSCHEDULE system view

Each row in the SYSSCHEDULE system view describes a time at which an event is to fire, as specified by the SCHEDULE clause of CREATE EVENT. The underlying system table for this view is ISYSSCHEDULE.

The SYSSCHEDULE view is a SQL Anywhere system view. See "SYSREMOTEUSER system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSSERVER system view

Each row in the SYSSERVER system view describes a remote server. The underlying system table for this view is ISYSSERVER.

The SYSSERVER view is a SQL Anywhere system view. See "SYSREMOTEUSER system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSSOURCE system view

Each row in the SYSSOURCE system view contains the source code, if applicable, for an object listed in the SYSOBJECT system view.

The SYSSOURCE view is a SQL Anywhere system view. See “SYSSOURCE system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSSQLSERVERTYPE system view

The SYSSQLSERVERTYPE system view contains information relating to compatibility with Adaptive Server Enterprise. The underlying system table for this view is ISYSSQLSERVERTYPE.

The SYSSQLSERVERTYPE view is a SQL Anywhere system view. See “SYSSQLSERVERTYPE system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSSUBPARTITIONKEY system view

This view is reserved for future use. Sybase IQ 15.1 does not support subpartitioning.

SYSSUBSCRIPTION system view

Each row in the SYSSUBSCRIPTION system view describes a subscription from one user ID (which must have REMOTE permissions) to one publication. The underlying system table for this view is ISYSSUBSCRIPTION.

The SYSSUBSCRIPTION view is a SQL Anywhere system view. See “SYSSUBSCRIPTION system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSSUBSCRIPTIONS consolidated view

Each row describes a subscription from one user ID (which must have REMOTE permissions) to one publication.

The SYSSUBSCRIPTIONS view is a SQL Anywhere consolidated view. See “SYSSUBSCRIPTIONS consolidated view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSSYNC system view

The SYSSYNC system view contains information relating to MobiLink synchronization. Some columns in this view contain potentially sensitive data. For that reason, access to this view is restricted to users with DBA authority. The SYSSYNC2 view provides public access to the data in this view except for the potentially sensitive columns. The underlying system table for this view is ISYSSYNC.

The SYSSYNC view is a SQL Anywhere system view. See “SYSSYNC system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSSYNC2 consolidated view

The SYSSYNC2 view provides public access to the data found in the SYSSYNC system view—information relating to MobiLink synchronization—without exposing potentially sensitive data.

The SYSSYNC2 view is a SQL Anywhere consolidated view. See "SYSSYNC2 consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSSYNCPUBLICATIONDEFAULTS consolidated view

The SYSSYNCPUBLICATIONDEFAULTS view provides the default synchronization settings associated with publications involved in MobiLink synchronization.

The SYSSYNCPUBLICATIONDEFAULTS view is a SQL Anywhere consolidated view. See "SYSSYNCPUBLICATIONDEFAULTS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSSYNCS consolidated view

The SYSSYNCS view contains information relating to MobiLink synchronization. Some columns in this view contain potentially sensitive data. For that reason, access to this view is restricted to users with DBA authority.

The SYSSYNCS view is a SQL Anywhere consolidated view. See "SYSSYNCS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSSYNCSCRIPT system view

Each row in the SYSSYNCSCRIPT system view identifies a stored procedure for MobiLink scripted upload. This view is almost identical to the SYSSYNCSCRIPTS view, except that the values in this view are in their raw format. To see them in their human-readable format, see "SYSSYNCSCRIPTS consolidated view".

The SYSSYNCSCRIPT view is a SQL Anywhere system view. See "SYSSYNCSCRIPT system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSSYNCSCRIPTS consolidated view

Each row in the SYSSYNCSCRIPTS view identifies a stored procedure for MobiLink scripted upload. This view is almost identical to the SYSSYNCSCRIPT system view, except that the values are in human-readable format, as opposed to raw data.

The SYSSYNCSCRIPTS view is a SQL Anywhere consolidated view. See "SYSSYNCSCRIPTS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSSYNCSUBSCRIPTIONS consolidated view

The SYSSYNCSUBSCRIPTIONS view contains the synchronization settings associated with MobiLink synchronization subscriptions.

The SYSSYNCSUBSCRIPTIONS view is a SQL Anywhere consolidated view. See "SYSSYNCSUBSCRIPTIONS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSSYNCUSERS consolidated view

A view of synchronization settings associated with MobiLink synchronization users.

The SYSSYNCUSERS view is a SQL Anywhere consolidated view. See "SYSSYNCUSERS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSTAB system view

Each row of the SYSTAB system view describes one table or view in the database. Additional information for views can be found in the SYSVIEW system view. The underlying system table for this view is ISYSTAB.

The SYSTAB view is a SQL Anywhere system view. See "SYSTAB system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSTABLE compatibility view (deprecated)

The SYSTABLE view is provided for compatibility with older versions of Sybase IQ that offered a SYSTABLE system table. However, the SYSTABLE system table has been replaced by the ISYSTAB system table, and its corresponding SYSTAB system view.

The SYSTABLE view is a SQL Anywhere compatibility view. See "SYSTABLE compatibility view (deprecated)" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views*.

SYSTABAUTH consolidated view

The SYSTABAUTH view contains information from the SYSTABLEPERM system view, but in a readable format.

The SYSTABAUTH view is a SQL Anywhere consolidated view. See "SYSTABAUTH consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSTABCOL system view

The SYSTABCOL system view contains one row for each column of each table and view in the database. The underlying system table for this view is ISYSTABCOL.

The SYSTABCOL view is a SQL Anywhere system view. See "SYSTABCOL system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSTABLEPERM system view

Permissions given by the GRANT statement are stored in the SYSTABLEPERM system view. Each row in this view corresponds to one table, one user ID granting the permission (grantor) and one user ID granted the permission (grantee). The underlying system table for this view is ISYSTABLEPERM.

The SYSTABLEPERM view is a SQL Anywhere system view. See "SYSTABLEPERM system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSTEXTCONFIG system view

Each row in the SYSTEXTCONFIG system view describes one text configuration object, for use with the full text search feature. The underlying system table for this view is ISYSTEXTCONFIG.

The SYSTEXTCONFIG view is a SQL Anywhere system view. See "SYSTEXTCONFIG system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSTEXTIDX system view

Each row in the SYSTEXTIDX system view describes one text index. The underlying system table for this view is ISYSTEXTIDX.

The SYSTEXTIDX view is a SQL Anywhere system view. See "SYSTEXTIDX system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSTEXTIDXTAB system view

Each row in the SYSTEXTIDXTAB system view describes a generated table that is part of a text index. The underlying system table for this view is ISYSTEXTIDXTAB.

The SYSTEXTIDXTAB view is a SQL Anywhere system view. See "SYSTEXTIDXTAB system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSTRIGGER system view

Each row in the SYSTRIGGER system view describes one trigger in the database. This view also contains triggers that are automatically created for foreign-key definitions that have a referential triggered action (such as ON DELETE CASCADE). The underlying system table for this view is ISYSTRIGGER.

The SYSTRIGGER view is a SQL Anywhere system view. See "SYSTRIGGER system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSTRIGGERS consolidated view

Each row in the SYSTRIGGERS view describes one trigger in the database. This view also contains triggers that are automatically created for foreign key definitions which have a referential triggered action (such as ON DELETE CASCADE).

The SYSTRIGGERS view is a SQL Anywhere consolidated view. See "SYSTRIGGERS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSTYPEMAP system view

The SYSTYPEMAP system view contains the compatibility mapping values for entries in the SYSSQLSERVERTYPE system view. The underlying system table for this view is ISYSTYPEMAP.

The SYSTYPEMAP view is a SQL Anywhere system view. See "SYSTYPEMAP system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSTYPES ASE compatibility view

This view is owned by user DBO. systypes contains one row for each system-supplied and user-defined datatype. Domains (defined by rules) and defaults are given, if they exist.You cannot alter the rows that describe system-supplied datatypes. See Table 8-1 on page 588

SYSUSER system view

Each row in the SYSUSER system view describes a user in the system. The underlying system table for this view is ISYSUSER.

The SYSUSER view is a SQL Anywhere system view. See “SYSUSER system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSUSERAUTH compatibility view (deprecated)

The SYSUSERAUTH view is provided for compatibility with older versions of Sybase IQ. Use the SYSUSERAUTHORITY system view instead.

The SYSUSERAUTH view is a SQL Anywhere compatibility view. See “SYSUSERAUTH compatibility view (deprecated)” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views*.

SYSUSERAUTHORITY system view

Each row of SYSUSERAUTHORITY system view describes an authority granted to one user ID. The underlying system table for this view is ISYSUSERAUTHORITY.

The SYSUSERAUTHORITY view is a SQL Anywhere system view. See “SYSUSERAUTHORITY system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSUSERLIST compatibility view (deprecated)

The SYSUSERLIST view is provided for compatibility with older versions of Sybase IQ. Each row of the SYSUSERLIST view describes a user, without exposing their user_id and password. Each user is identified by their user name.

The SYSUSERLIST view is a SQL Anywhere compatibility view. See "SYSUSERLIST compatibility view (deprecated)" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views*.

SYSUSERMESSAGE system view

Each row in the SYSUSERMESSAGE system view holds a user-defined message for an error condition. The underlying system table for this view is ISYSUSERMESSAGE.

The SYSUSERMESSAGE view is a SQL Anywhere system view. See "SYSUSERMESSAGE system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSUSEROPTIONS consolidated view

The SYSUSEROPTIONS view contains the option settings that are in effect for each user. If a user has no setting for an option, this view displays the public setting for the option.

The SYSUSEROPTIONS view is a SQL Anywhere consolidated view. See "SYSUSEROPTIONS consolidated view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSUSERPERM compatibility view (deprecated)

This view is deprecated because it only shows the authorities and permissions available in previous versions. Change your application to use the SYSUSERAUTHORITY system view instead. Each row of the SYSUSERPERM view describes one user ID.

The SYSUSERPERM view is a SQL Anywhere compatibility view. See "SYSUSERPERM compatibility view (deprecated)" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views*.

SYSUSERPERMS compatibility view (deprecated)

This view is deprecated because it only shows the authorities and permissions available in previous versions. Change your application to use the SYSUSERAUTHORITY system view instead. Similar to the SYSUSERPERM view, each row of the SYSUSERPERMS view describes one user ID. However, password information is not included. All users are allowed to read from this view.

The SYSUSERPERMS view is a SQL Anywhere compatibility view. See "SYSUSERPERMS compatibility view (deprecated)" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views*.

SYSUSERTYPE system view

Each row in the SYSUSERTYPE system view holds a description of a user-defined data type. The underlying system table for this view is ISYSUSERTYPE.

The SYSUSERTYPE view is a SQL Anywhere system view. See "SYSUSERTYPE system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSUSERS ASE compatibility view

This view is owned by user DBO. sysusers contains one row for each user allowed in the database, and one row for each group or roles. See Table 8-1 on page 588

SYSVIEW system view

Each row in the SYSVIEW system view describes a view in the database. You can find additional information about views in the SYSTAB system view. The underlying system table for this view is ISYSVIEW.

You can also use the sa_materialized_view_info system procedure for a readable format of the information for materialized views.

The SYSVIEW view is a SQL Anywhere system view. See “SYSVIEW system view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

SYSVIEWS consolidated view

Each row of the SYSVIEWS view describes one view, including its view definition.

The SYSVIEWS view is a SQL Anywhere consolidated view. See “SYSVIEWS consolidated view” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views*.

SYSWEBSERVICE system view

Each row in the SYSWEBSERVICE system view holds a description of a web service.

The SYSWEBSERVICE view is a SQL Anywhere system view. See "SYSWEBSERVICE system view" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views*.

Transact-SQL compatibility views

Adaptive Server Enterprise and Sybase IQ have different system catalogs, reflecting the different uses for the two products.

In Adaptive Server Enterprise, there is a single master database containing a set of system tables holding information that applies to all databases on the server. Many databases may exist within the master database, and each has additional system tables associated with it.

In Sybase IQ, each database exists independently, and contains its own system tables. There is no master database that contains system information on a collection of databases. Each server may run several databases at a time, dynamically loading and unloading each database as needed.

The Adaptive Server Enterprise and Sybase IQ system catalogs are different. The Adaptive Server Enterprise system tables and views are owned by the special user dbo, and exist partly in the master database, partly in the sybsecurity database, and partly in each individual database; the Sybase IQ system tables and views are owned by the special user SYS and exist separately in each database.

To assist in preparing compatible applications, Sybase IQ provides a set of views owned by the special user dbo, which correspond to the Adaptive Server Enterprise system tables and views. Where architectural differences make the contents of a particular Adaptive Server Enterprise table or view meaningless in a Sybase IQ context, the view is empty, containing only the column names and data types.

Table 8-1, Table 8-2, and Table 8-3 list the Adaptive Server Enterprise system tables and their implementation in the Sybase IQ system catalog. The owner of all tables is dbo in each DBMS.

Tables in each Adaptive Server Enterprise database

Table 8-1: Tables in each ASE database

Table name	Description	Data?	Supported by IQ?
sysalternates	One row for each user mapped to a database user	No	No
syscolumns	One row for each column in a table or view, and for each parameter in a procedure. In Sybase IQ, use the owner name dbo when querying, i.e. dbo.syscolumns.	Yes	Yes
syscomments	One or more rows for each view, rule, default, and procedure, giving SQL definition statement.	Yes	Yes
sysconstraints	One row for each referential and check constraint associated with a table or column.	No	No
sysdepends	One row for each procedure, view, or table that is referenced by a procedure, view.	No	No
sysindexes	One row for each clustered or nonclustered index, and one row for each table with no indexes, and an additional row for each table containing text or image data. In Sybase IQ, use the owner name dbo when querying, i.e. dbo.sysindexes.	Yes	Yes
sysiqobjects	One row for each system table, user table, view, procedure, trigger, event, join index, constraint, domain (sysdomain), domain (sysusertype), column, and index.	Yes	Yes
sysiqvindex	One row for each non-fp iq index.	Yes	Yes
syskeys	One row for each primary, foreign, or common key; set by user (not maintained by Adaptive Server Enterprise).	No	No
syslogs	Transaction log.	No	No
sysobjects	One row for each table, view, procedure, rule, default, log, and (in tempdb only) temporary object.	Contains compatible data only	Yes

Table name	Description	Data?	Supported by IQ?
sysprocedures	One row for each view, rule, default, and procedure, giving internal definition.	No	No
sysprotects	User permissions information.	No	No
sysreferences	One row for each referential integrity constraint declared on a table or column.	No	No
sysroles	Maps server-wide roles to local database groups.	No	No
syssegments	One row for each segment (named collection of disk pieces).	No	No
systhresholds	One row for each threshold defined for the database.	No	No
systypes	One row for each system-supplied and user-defined data type.	Yes	Yes
sysusers	One row for each user allowed in the database.	Yes	Yes

Tables in the Adaptive Server Enterprise master database

Table 8-2: ASE master database tables

Table name	Description	Data?	Supported by IQ?
syscharsets	One row for each character set or sort order	No	No
sysconfigures	One row for each configuration parameter that can be set by a user	No	No
syscurconfigs	Information about configuration parameters currently being used by the server	No	No
sysdatabases	One row for each database on the server	No	No
sysdevices	One row for each tape dump device, disk dump device, disk for databases, and disk partition for databases	No	No
sysengines	One row for each server currently online	No	No
syslanguages	One row for each language (except U.S. English) known to the server	No	No
syslocks	Information about active locks	No	No
sysloginroles	One row for each server login that possesses a system-defined role	No	No
syslogins	One row for each valid user account	Yes	Yes
sysmessages	One row for each system error or warning	No	No
sysprocesses	Information about server processes	No	No
sysremotelogins	One row for each remote user	No	No
syssrvroles	One row for each server-wide role	No	No
sysservers	One row for each remote server	No	No
sysusages	One row for each disk piece allocated to a database	No	No

Tables in the Adaptive Server Enterprise sybsecurity database

Table 8-3: ASE sybsecurity database tables

Table name	Description	Data?	Supported by IQ?
sysaudits	One row for each audit record	No	No
sysauditoptions	One row for each global audit option	No	No

CHAPTER 9

System Tables

About this chapter

The structure of every Sybase IQ database is described in a number of system tables.

The DUMMYsystem table is the only system table you are permitted to access directly. For all other system tables, you access their underlying data via their corresponding views. See Chapter 8, “System Views.”

Contents

System table list

Table 9-1 lists all Sybase IQ system tables.

Table 9-1: List of system tables

System table	Internal use only?
DUMMY	No
ISYSARTICLE	Yes
ISYSARTICLECOL	Yes
ISYSATTRIBUTE	Yes
ISYSATTRIBUTENAME	Yes
ISYSCAPABILITY	Yes
ISYSCHECK	Yes
ISYSCOLPERM	Yes
ISYSCOLSTAT	Yes
ISYSCONSTRAINT	Yes
ISYSDBFILE	Yes
ISYSDBSPACE	Yes
ISYSDBSPACEPERM	Yes
ISYSDEPENDENCY	Yes
ISYSDOMAIN	Yes
ISYSEVENT	Yes
ISYSEXTERNENV	Yes
ISYSEXTERNENVOBJECT	Yes
ISYSEXTERNLOGIN	Yes
ISYSFKEY	Yes
ISYSGROUP	Yes
ISYSHISTORY	Yes
ISYSIDX	Yes
ISYSIDXCOL	Yes
ISYSIQBACKUPHISTORY	Yes
ISYSIQBACKUPHISTORYDETAIL	Yes
ISYSIQDBFILE	Yes
ISYSIQDBSPACE	Yes
ISYSIQIDX	Yes
ISYSIQINFO	Yes
ISYSIQJOINIDX	Yes
ISYSIQJOINIXCOLUMN	Yes
ISYSIQMPXLOGINPOLICYOPTION	Yes
ISYSIQMPXSERVER	Yes
ISYSIQPARTITIONCOLUMN	Yes
ISYSIQTAB	Yes

System table	Internal use only?
ISYSIQTABCOL	Yes
ISYSJAR	Yes
ISYSJARCOMPONENT	Yes
ISYSJAVACLASS	Yes
ISYSLOGINMAP	Yes
ISYSLOGINPOLICY	Yes
ISYSLOGINPOLICYOPTION	Yes
ISYSMVOPTION	Yes
ISYSMVOPTIONNAME	Yes
ISYSOBJECT	Yes
ISYSOPTION	Yes
ISYSOPTSTAT	Yes
ISYSPARTITION	Yes
ISYSPARTITIONKEY	Yes
ISYSPARTITIONSCHEME	Yes
ISYSPHYSIDX	Yes
ISYSPROCEDURE	Yes
ISYSPROCPARM	Yes
ISYSPROCPERM	Yes
ISYSPROXYTAB	Yes
ISYSPUBLICATION	Yes
ISYSREMARK	Yes
ISYSREMOTEOPTION	Yes
ISYSREMOTEOPTIONTYPE	Yes
ISYSREMOTETYPE	Yes
ISYSREMOTEUSER	Yes
ISYSSCHEDULE	Yes
ISYSSERVER	Yes
ISYSSOURCE	Yes
ISYSSQLSERVERTYPE	Yes
ISYSSUBPARTITIONKEY	Yes
ISYSSUBSCRIPTION	Yes
ISYSSYNC	Yes
ISYSSYNCPROFILE	Yes
ISYSSYNCSCRIPT	Yes
ISYSTAB	Yes
ISYSTABCOL	Yes

System table	Internal use only?
ISYSTABLEPERM	Yes
ISYSTEXTCONFIG	Yes
ISYSTEXTIDX	Yes
ISYSTEXTIDXTAB	Yes
ISYSTRIGGER	Yes
ISYSTYPEMAP	Yes
ISYSUSER	Yes
ISYSUSERAUTHORITY	Yes
ISYSUSERMESSAGE	Yes
ISYSUSERTYPE	Yes
ISYSVIEW	Yes
ISYSWEBSERVICE	Yes

DUMMY system table

The DUMMY table is provided as a table that always has exactly one row. This can be useful for extracting information from the database, as in the following example that gets the current user ID and the current date from the database.

```
SELECT USER, today(*) FROM SYS.DUMMY
```

The DUMMY table is a SQL Anywhere system table. For detailed information, including a link to its SQL Anywhere system view see "DUMMY system table" in the SQL Anywhere documentation at *SQL Anywhere 11.0.1* > *SQL Anywhere Server - SQL Reference* > *System Objects* > *Tables* > *System tables*.

APPENDIX A

Compatibility with Other Sybase Databases

About this appendix

The information in this appendix is intended to simplify migration to Sybase IQ from other Sybase databases, and to serve as a guide for creating Sybase IQ applications that are compatible with Adaptive Server Enterprise or SQL Anywhere. Beginning with an overview of Transact-SQL, it compares these databases in several areas to be aware of when you are moving to Sybase IQ:

- Architecture
- Data types
- Data definition language
- Data manipulation language
- Stored procedure language

Compatibility features are addressed in each new version of Sybase IQ. This appendix compares Sybase IQ 15.1 with Adaptive Server Enterprise 15.0.3 (and earlier releases), and SQL Anywhere 11.0.1.

Topics

Compatibility information elsewhere in this book

You can find additional compatibility information in the following chapters:

- Chapter 2, “Database Options,” in *Reference: Statements and Options*. See Transact-SQL compatibility options.
- Chapter 3, “SQL Data Types.” See compatibility information for each data type; also see Data type conversions on page 89.
- *Reference: Statements and Options*. See the compatibility information in each command.

A note about SQL Anywhere

Sybase IQ is an extension of SQL Anywhere. In most cases, SQL syntax, functions, options, utilities, procedures, and other features are common to both products. There are, however, important differences. Do not assume that features described in SQL Anywhere documentation are supported for Sybase IQ.

The Sybase IQ documentation set calls many, but not all differences. Sybase IQ documentation always supersedes the SQL Anywhere documentation. Except for topics where the Sybase IQ documentation refers you to SQL Anywhere documentation, always refer to the documentation listed as “Documentation for Sybase IQ” in “About This Book,” immediately after the Table of Contents in each Sybase IQ book.

An overview of Transact-SQL support

Sybase IQ, like SQL Anywhere, supports a large subset of **Transact-SQL**, which is the dialect of SQL supported by Sybase Adaptive Server Enterprise.

The goal of Transact-SQL support in Sybase IQ is to provide application portability. Many applications, stored procedures, and batch files can be written for use with both Adaptive Server Enterprise and Sybase IQ databases.

The aim is to write applications to work with both Adaptive Server Enterprise and Sybase IQ. Existing Adaptive Server Enterprise applications generally require some changes to run on SQL Anywhere or Sybase IQ databases.

Transact-SQL support in Sybase IQ takes the following form:

- Most SQL statements are compatible between Sybase IQ and Adaptive Server Enterprise.
- For some statements, particularly in the procedure language used in procedures and batches, a separate Transact-SQL statement is supported along with the syntax supported in earlier versions of Sybase IQ. For these statements, SQL Anywhere and Sybase IQ support two dialects of SQL. In this appendix, we name those dialects Transact-SQL and Watcom-SQL.
- A procedure or batch is executed in either the Transact-SQL or Watcom-SQL dialect. You must use control statements from one dialect only throughout the batch or procedure. For example, each dialect has different flow control statements.

Similarities and differences

Sybase IQ supports a high percentage of Transact-SQL language elements, functions, and statements for working with existing data.

Further, Sybase IQ supports a very high percentage of the Transact-SQL stored procedure language (CREATE PROCEDURE syntax, control statements, and so on), and many, but not all, aspects of Transact-SQL data definition language statements.

There are design differences in the architectural and configuration facilities supported by each product. Device management, user management, and maintenance tasks such as backups tend to be system-specific. Even here, however, Sybase IQ provides Transact-SQL system tables as views, where the tables that are not meaningful in Sybase IQ have no rows. Also, Sybase IQ provides a set of system procedures for some of the more common administrative tasks.

Adaptive Server architectures

Adaptive Server Enterprise, SQL Anywhere, and Sybase IQ are complementary products, with architectures designed to suit their distinct purposes. Sybase IQ is a high-performance decision-support server designed specifically for data warehousing and analytic processing. SQL Anywhere works well as a workgroup or departmental server requiring little administration, and as a personal database. Adaptive Server Enterprise works well as an enterprise-level server for large databases, with a focus on transaction processing.

This section describes architectural differences among the three products. It also describes the Adaptive Server Enterprise-like tools that Sybase IQ and SQL Anywhere include for compatible database management.

Servers and databases

The relationship between servers and databases is different in Adaptive Server Enterprise from Sybase IQ and SQL Anywhere.

In Adaptive Server Enterprise, each database exists inside a server, and each server can contain several databases. Users can have login rights to the server, and can connect to the server. They can then connect to any of the databases on that server, provided that they have permissions. System-wide system tables, held in a master database, contain information common to all databases on the server.

In Sybase IQ, there is nothing equivalent to the Adaptive Server Enterprise master database. Instead, each database is an independent entity, containing all of its system tables. Users can have connection rights to a database, rather than to the server. When a user connects, he or she connects to an individual database. There is no system-wide set of system tables maintained at a master database level. Each Sybase IQ database server can dynamically start and stop a database, to which users can maintain independent connections. Sybase strongly recommends that you run only one Sybase IQ database per server.

SQL Anywhere and Sybase IQ provide tools in their Transact-SQL support and Open Server support to allow some tasks to be carried out in a manner similar to Adaptive Server Enterprise. There are differences, however, in exactly how these tools are implemented.

See the *System Administration Guide: Volume 2*, Chapter 3, “Sybase IQ as a Data Server,” for details on how to use isql to connect to a specific database on a server with multiple databases.

Space allocation and device management

All three products use different models for managing devices and allocating disk space initially and later, reflecting the different uses for the products. For example:

- In Adaptive Server Enterprise, you allocate space in database devices initially using DISK INIT and then create a database on one or more database devices. You can add more space using ALTER DATABASE or automatically, using thresholds.
- In Sybase IQ, you initially allocate space by listing raw devices in the CREATE DATABASE statement. You can add more space manually using CREATE DBSPACE. Although you cannot add space automatically, you can create events to warn the DBA before space is actually needed. Sybase IQ can also use file system space. Sybase IQ does not support Transact-SQL DISK statements, such as DISK INIT, DISK MIRROR, DISK REFIT, DISK REINIT, DISK REMIRROR, and DISK UNMIRROR.
- SQL Anywhere is similar to Sybase IQ, except that the initial CREATE DATABASE statement takes a single file system file instead of a list of raw devices. SQL Anywhere also lets you initialize its databases using a command utility dbinit, which Sybase IQ does not support.

For information on disk management, see the *System Administration Guide: Volume 1*.

System tables, catalog store, and IQ store

An IQ database is a joint data store consisting of:

- The catalog store includes system tables and stored procedures, and resides in a set of tables that are compatible with SQL Anywhere.
- The permanent IQ store is the set of Sybase IQ tables. Table data is stored in indexes.
- The temporary store consists of a set of temporary tables which the database server uses for sorting and other temporary processing.

Catalog distinctions and compatibility features include:

- SQL Anywhere and Sybase IQ use a different schema from Adaptive Server Enterprise for the catalog (tables, columns, and so on).

- SQL Anywhere and Sybase IQ provide compatibility views that mimic relevant parts of the Adaptive Server Enterprise system tables, although there are performance implications when using them. For a list and individual descriptions, see Chapter 8, “System Views” and Chapter 9, “System Tables.” For a complete list of SQL Anywhere compatibility views, see *SQL Anywhere Server – SQL Reference*.
- In Adaptive Server Enterprise, the database owner (user ID dbo) owns the catalog objects.
- In SQL Anywhere and Sybase IQ, the system owner (user ID SYS) owns the catalog objects.

Note A dbo user ID owns the Adaptive Server Enterprise-compatible system views provided by Sybase IQ. The dba user ID owns a small number of Sybase IQ system tables, used for Sybase IQ user and multiplex administration.

Administrative roles

Adaptive Server Enterprise has a more elaborate set of administrative roles than either SQL Anywhere or Sybase IQ. In Adaptive Server Enterprise, there is a set of distinct roles, although more than one login account on an Adaptive Server Enterprise can be granted any role, and one account can possess more than one role.

In Adaptive Server Enterprise, distinct roles include:

- **System administrator** Responsible for general administrative tasks unrelated to specific applications; can access any database object.
- **System security officer** Responsible for security-sensitive tasks in Adaptive Server Enterprise, but has no special permissions on database objects.
- **Database owner** Has full permissions on objects inside the database he or she owns, can add users to a database and grant other users the permission to create objects and execute commands within the database.
- **Data definition statements** Permissions can be granted to users for specific data definition statements, such as CREATE TABLE or CREATE VIEW, enabling the user to create database objects.

- **Object owner**   Each database object has an owner who may grant permissions to other users to access the object. The owner of an object automatically has all permissions on the object.

In SQL Anywhere and Sybase IQ, the following database-wide permissions have administrative roles:

- **Database Administrator (DBA authority)**   Has, like the Adaptive Server Enterprise Database Owner, full permissions on all objects inside the database (other than objects owned by SYS) and can grant other users the permission to create objects and execute commands within the database. The default database administrator is user ID DBA.
- **RESOURCE permission**   Allows a user to create any kind of object within a database. This is in place of the Adaptive Server Enterprise scheme of granting permissions on individual CREATE statements.
- **Object owner**   Sybase IQ has object owners in the same way Adaptive Server Enterprise does. The owner of an object automatically has all permissions on the object, including the right to grant permissions.

For seamless access to data held in both Adaptive Server Enterprise and Sybase IQ, create user IDs with appropriate permissions in the database (RESOURCE in Sybase IQ, or permission on individual CREATE statements in Adaptive Server Enterprise) and create objects from that user ID. If you use the same user ID in each environment, object names and qualifiers can be identical in the two databases, providing compatible access.

Data types

This section discusses compatibility information for data types. For details of Sybase IQ data types, see Chapter 3, “SQL Data Types.”

Note Data types that are not included in this section are currently supported by all three products.

Bit data type

All three products support the BIT data type, with these differences:

- SQL Anywhere permits only 0 or 1.
- Adaptive Server Enterprise and Sybase IQ implicitly convert integral data types to BIT. Nonzero values are stored as 1 (TRUE).

Character data types

All three products permit CHAR and VARCHAR data, but each product treats these types differently.

- SQL Anywhere treats all strings as VARCHAR, even in a blank-padded database.
- Adaptive Server Enterprise and Sybase IQ differentiate between CHAR (fixed-length) and VARCHAR (variable-length) data.

  Adaptive Server Enterprise trims trailing blank spaces from VARCHAR values. Sybase IQ trims trailing blanks from VARCHAR values depending on the form of the data and the operation. See "Character data types" in Chapter 3, "SQL Data Types."

When inserting into CHAR or VARCHAR:

- SQL Anywhere permits inserting integral data types into CHAR or VARCHAR (implicit conversion).
- Adaptive Server Enterprise and Sybase IQ require explicit conversion.

The maximum size of a column is determined as follows:

- Adaptive Server Enterprise CHAR and VARCHAR depend on the logical page size, which can be 2K, 4K, 8K, and 16K. For example:
  - 2K page size allows a column as large as a single row, about 1962 bytes.
  - 4K page size allows a column as large as about 4010 bytes.
- SQL Anywhere supports up to 32K-1 with CHAR and VARCHAR, and up to 2GB with LONG VARCHAR.
- SQL Anywhere supports the name LONG VARCHAR and its synonym TEXT, while Adaptive Server Enterprise supports only the name TEXT, not the name LONG VARCHAR.

- Sybase IQ supports CHAR and VARCHAR up to 32K-1 bytes.

  Sybase IQ also supports up to 512TB (with an IQ page size of 128KB) and 2PB (with an IQ page size of 512KB) with LONG VARCHAR. For information on the LONG VARCHAR data type in Sybase IQ, see *Large Objects Management in Sybase IQ*.
- Adaptive Server Enterprise supports NCHAR, NVARCHAR, UNICHAR, UNIVARCHAR data types. N is for multibyte character sets; UNI is for single-byte character sets.
- SQL Anywhere and Sybase IQ support Unicode in the CHAR and VARCHAR data types, rather than as a separate data type.
- For compatibility between Sybase IQ and Adaptive Server Enterprise, always specify a length for character data types.

Binary data types

The following table summarizes binary data type support.

Table A-1: Binary data type supported sizes

Data type	Adaptive Server Enterprise	SQL Anywhere	Sybase IQ
BINARY	< page size	32KB - 1	255
VARBINARY	< page size	32KB - 1	32KB - 1
LONG BINARY*	not supported	2GB - 1	512TB (IQ page size 128KB) 2PB (IQ page size 512KB)
IMAGE	2GB	2GB - 1	use LONG BINARY*

*For information on the LONG BINARY data type in Sybase IQ, see *Large Objects Management in Sybase IQ*. This feature requires a separate license.

Adaptive Server Enterprise and SQL Anywhere display binary data differently when projected:

- Sybase IQ supports both Adaptive Server Enterprise and SQL Anywhere display formats.
- If ‘123’ is entered in a BINARY field the SQL Anywhere display format is by bytes, as ‘123’; the Adaptive Server Enterprise display format is by nibbles, as ‘0x616263’.

Date, time, datetime, and timestamp data types

Although all three products support some form of date and time data, there are some differences.

- SQL Anywhere and Sybase IQ support the 4-byte date and time data types.
- Adaptive Server Enterprise supports an 8-byte datetime type, and timestamp as a user-defined data type (domain) implemented as binary (8).
- SQL Anywhere and Sybase IQ support an 8-byte timestamp type, and an 8-byte datetime domain implemented as timestamp. The millisecond precision of the Anywhere/Sybase IQ datetime data type differs from that of Adaptive Server Enterprise.

Display formats for dates have different defaults:

- Adaptive Server Enterprise defaults to displaying dates in the format "MMM-DD-YYYY" but can be changed by setting an option.
- SQL Anywhere and Sybase IQ default to the ISO "YYYY-MM-DD" format but can be changed by setting an option.

Time conversions are as follows:

- Adaptive Server Enterprise varies the way it converts time stored in a string to an internal time, depending on whether the fraction part of the second was delimited by a colon or a period.
- SQL Anywhere and Sybase IQ convert times in the same way, regardless of the delimiter.

When you insert a time into a DATETIME column:

- Adaptive Server Enterprise and Sybase IQ default to supplying 1st January 1900.
- SQL Anywhere defaults to supplying the current date.

TIME and DATETIME values retrieved from an Adaptive Server Enterprise database change when inserted into a Sybase IQ table with a DATETIME column using INSERT...LOCATION. The INSERT...LOCATION statement uses Open Client, which has a DATETIME precision of 1/300 of a second.

For example, assume that the following value is stored in a table column in an Adaptive Server Enterprise database:

```
2004-11-08 10:37:22.823
```

When you retrieve and store it in a Sybase IQ table using INSERT...LOCATION, the value becomes:

```
2004-11-08 10:37:22.823333
```

Compatibility of datetime and time values from ASE

A DATETIME or TIME value retrieved from an Adaptive Server Enterprise database using INSERT...LOCATION can have a different value due to the datetime precision of Open Client.

For example, the DATETIME value in the Adaptive Server Enterprise database is ‘2004-11-08 10:37:22.823’ as retrieved using INSERT...LOCATION is ‘2004-11-08 10:37:22.823333’.

Numeric data types

Adaptive Server Enterprise, SQL Anywhere, and Sybase IQ have different default precision and scale:

- In Adaptive Server Enterprise, the default is precision 18 scale 0.
- In SQL Anywhere, the default is precision 30 scale 6.
- In Sybase IQ, the default is precision 126 scale 38. Because these defaults are too large for TDS and for some client tools, always specify a precision and scale for Sybase IQ exact numeric types.

Approximate numeric data types

Adaptive Server Enterprise differs from SQL Anywhere and Sybase IQ in how the FLOAT(precision) data type is interpreted: that is, when to create a 4-byte data type and when to create an 8-byte data type.

REAL values are four bytes, DOUBLE values are eight bytes. Turning on the FLOAT_AS_DOUBLE option makes the FLOAT keyword behave like Adaptive Server Enterprise's FLOAT keyword when a precision is not specified. Since Adaptive Server Enterprise treats its own FLOAT values as DOUBLE, enabling the option makes Sybase IQ to treat FLOAT values in the same way Adaptive Server Enterprise treats FLOAT values.

Turning off the FLOAT_AS_DOUBLE option makes the FLOAT keyword behave like a Sybase IQ and SQL Anywhere four byte REAL value.

For Adaptive Server Enterprise, the precision specified in FLOAT(precision) means decimal precision. For SQL Anywhere and Sybase IQ, precision is an integer expression that specifies the number of bits used in the mantissa of the floating point number.

For more information on the Sybase IQ FLOAT data type, see "Numeric data types" in Chapter 3, "SQL Data Types."

Text data type

Support for TEXT data is currently implemented as follows:

- Adaptive Server Enterprise and SQL Anywhere support up to 2GB with LONG VARBINARY and TEXT.
- Sybase IQ supports up to 32KB - 1 with VARCHAR. Sybase IQ also supports up to 512TB (with an IQ page size of 128KB) and 2PB (with an IQ page size of 512KB) with LONG VARCHAR. For information on the LONG VARCHAR data type in Sybase IQ, see *Large Objects Management in Sybase IQ*.

Image data type

Support for IMAGE data is currently implemented as follows:

- Adaptive Server Enterprise and SQL Anywhere support up to 2GB with IMAGE.
- Sybase IQ supports up to 512TB (with an IQ page size of 128KB) and 2PB (with an IQ page size of 512KB) with LONG BINARY. For information on the LONG BINARY data type in Sybase IQ, see *Large Objects Management in Sybase IQ*.

Java data types

Adaptive Server Enterprise allows Java data types in the database. SQL Anywhere and Sybase IQ do not.

Data definition language

This section discusses compatibility information for creating database objects. See also “System tables, catalog store, and IQ store” on page 599 and “Space allocation and device management” on page 598 for related information.

Creating a Transact-SQL compatible database

This section describes choices you must make when creating or rebuilding a database.

Here are the basic steps you need to take to create a Transact-SQL compatible database. The remainder of the section describes which options you need to set.

❖ **Creating a Transact-SQL compatible database from Sybase Central**

1. One page of the Create Database wizard is named Default Database Attributes.
2. To emulate Adaptive Server Enterprise, choose “Emulate Adaptive Server Enterprise” which automatically selects case-sensitivity for string comparisons and case-sensitivity for passwords.

❖ **Creating a Transact-SQL compatible database using the CREATE DATABASE statement**

- Type the following statement, for example, in Interactive SQL:

```
CREATE DATABASE 'db-name.db'
CASE RESPECT BLANK PADDING ON
```

Case-sensitivity

Case-sensitivity in databases refers to:

- **Data** The case-sensitivity of the data is reflected in indexes, in the results of queries, and so on.
- **Identifiers** Identifiers include table names, column names, user IDs, and so on.
- **Passwords** Case-sensitivity of passwords is treated differently from other identifiers.

Case-sensitivity of data

You decide the case-sensitivity of Sybase IQ data in comparisons when you create the database. By default, Sybase IQ databases are case-sensitive in comparisons, although data is always held in the case in which you enter it.

Adaptive Server Enterprise sensitivity to case depends on the sort order installed on the Adaptive Server Enterprise system. You can change case-sensitivity for single-byte character sets by reconfiguring the Adaptive Server Enterprise sort order.

Case-sensitivity of identifiers

Sybase IQ does not support case-sensitive identifiers. In Adaptive Server Enterprise, the case-sensitivity of identifiers follows the case-sensitivity of the data.

In Adaptive Server Enterprise, user-defined data type names are case-sensitive. In Sybase IQ, they are case-insensitive.

User IDs and passwords

In Sybase IQ and SQL Anywhere, all passwords in newly-created databases are case-sensitive, regardless of the case-sensitivity of the database. The default user ID is DBA and the password for this user is lowercase **sql**.

When you rebuild an existing database, Sybase IQ and SQL Anywhere determine the case-sensitivity of the password as follows:

- If the database was originally entered in a case-insensitive database, the password remains case-insensitive.
- If the password was originally entered in a case-sensitive database, uppercase and mixed-case passwords remain case-sensitive. If the password was entered in all lowercase, then the password becomes case-insensitive.
- Changes to both existing passwords and new passwords are case-sensitive.

In Adaptive Server Enterprise, the case-sensitivity of user IDs and passwords follows the case-sensitivity of the server.

Ensuring compatible object names

Each database object must have a unique name within a certain **name space**. Outside this name space, duplicate names are allowed. Some database objects occupy different name spaces in Adaptive Server Enterprise as compared to SQL Anywhere and Sybase IQ.

Table name uniqueness

Table name uniqueness requirements apply within a database:

- For Sybase IQ and SQL Anywhere, table names must be unique within a database for a given owner. For example, both user1 and user2 can create a table called employee; uniqueness is provided by the fully qualified names, user1.employee and user2.employee.
- For Adaptive Server Enterprise, table names must be unique within the database and to the owner.

Index name uniqueness

Index name uniqueness requirements apply within a table. In all three products, indexes are owned by the owner of the table on which they are created. Index names must be unique on a given table, but any two tables can have an index of the same name, even for the same owner. For example, in all three products, tables t1 and t2 can have indexes of the same name, whether they are owned by the same or different users.

Renaming indexes and foreign keys

Sybase IQ allows you to rename explicitly created indexes, foreign key role names of indexes, and foreign keys, using the ALTER INDEX statement. Adaptive Server Anywhere allows you to rename indexes, foreign key role names, and foreign keys, using the ALTER INDEX statement. Adaptive Server Enterprise does not allow you to rename these objects.

CREATE TABLE statement

When creating tables for compatibility, be aware of the following items.

NULL in columns

For compatible treatment of NULL:

- SQL Anywhere and Sybase IQ assume that columns can be null unless NOT NULL is stated in the column definition. You can change this behavior by setting the database option ALLOW_NULLS_BY_DEFAULT to the Transact-SQL compatible setting of OFF.
- SQL Anywhere assumes that BIT columns only cannot be NULL.
- Adaptive Server Enterprise assumes that columns cannot be null unless NULL is stated.

Check constraints

Sybase IQ enforces check constraints on base, global temporary, and local temporary tables, and on user-defined data types. Users can log check integrity constraint violations and specify the number of violations that can occur before a LOAD statement rolls back.

Sybase IQ does not allow the creation of a check constraint that it cannot evaluate, such as those composed of user-defined functions, proxy tables, or non-Sybase IQ tables. Constraints that cannot be evaluated are detected the first time the table on which the check constraint is defined is used in a LOAD, INSERT, or UPDATE statement. Sybase IQ does not allow check constraints containing:

- Subqueries
- Expressions specifying a host language parameter, a SQL parameter, or a column as the target for a data value
- Set functions
- Invocations of nondeterministic functions or functions that modify data

Adaptive Server Enterprise and SQL Anywhere enforce CHECK constraints. SQL Anywhere allows subqueries in check constraints.

Sybase IQ supports user-defined data types that allow constraints to be encapsulated in the data type definition.

Referential integrity constraints

Sybase IQ enforces referential integrity as described in Chapter 9, “Ensuring Data Integrity” in the *System Administration Guide: Volume 1*.

Actions for enforcing integrity are supported as follows:

- SQL Anywhere supports all ANSI actions: SET NULL, CASCADE, DEFAULT, RESTRICT.
- Adaptive Server Enterprise supports two of these actions: SET NULL, DEFAULT.

Note You can achieve CASCADE in Adaptive Server Enterprise by using triggers instead of referential integrity.

- Sybase IQ supports the RESTRICT action only.
- Sybase IQ does not support NOT NULL FOREIGN KEY.
- Sybase IQ has the restriction that a column cannot be both a candidate key and a foreign key at the same time.

Default values in a column

Default value support differs as follows:

- Adaptive Server Enterprise and SQL Anywhere support specifying a default value for a column.
- Only SQL Anywhere supports DEFAULT UTC TIMESTAMP.

- Sybase IQ supports specifying a default value for a column, except for the special values DEFAULT UTC TIMESTAMP and DEFAULT CURRENT UTC TIMESTAMP. Sybase IQ also ignores settings for the DEFAULT_TIMESTAMP_INCREMENT database option.

Identity columns

Identity column support differs as follows:

- Sybase IQ supports IDENTITY or DEFAULT AUTOINCREMENT as a default value. Sybase IQ supports identity columns of any numeric type with any precision and scale 0, and the column can be NULL. Sybase IQ identity columns must be positive and are limited by the range of the data type. Sybase IQ supports a single identity column per table, and requires database option IDENTITY_INSERT set to a table name for explicit inserts and updates. To drop a table with an IDENTITY column, you cannot have IDENTITY_INSERT set to that table. The table can contain data when adding an identity column. Tables derived using SELECT INTO do not have Identity/Autoincrement columns. Sybase IQ views cannot contain IDENTITY/DEFAULT AUTOINCREMENT columns.
- SQL Anywhere supports the AUTOINCREMENT default value. SQL Anywhere supports identity columns of any numeric type with any allowable scale and precision. The identity column value can be positive, negative, or zero, limited by the range of the data type. SQL Anywhere supports any number of identity columns per table, and does not require identity_insert for explicit inserts, drops, and updates. The table must be empty when adding identity columns. SQL Anywhere identity columns can be altered to be nonidentity columns, and vice versa. You can add or drop AUTOINCREMENT columns from SQL Anywhere views.
- Adaptive Server Enterprise supports a single identity column per table. ASE identity columns are restricted to only numeric data type scale 0, maximum precision 38. They must be positive, are limited by the range of the data type, and cannot be null. Adaptive Server Enterprise requires identity_insert for explicit inserts and drops, but not for updates to the identity column. The table can contain data when you add an identity column. ASE users cannot explicitly set the next value chosen for an identity column. ASE views cannot contain IDENTITY/AUTOINCREMENT columns. When using SELECT INTO under certain conditions, ASE allows Identity/Autoincrement columns in the result table if they were in the table being selected from.

Computed columns

Computed column support differs as follows:

- SQL Anywhere supports computed columns that can be indexed.
- Adaptive Server Enterprise and Sybase IQ do not.

Temporary tables

You can create a temporary table by placing a pound sign (#) without an owner specification in front of the table name in a CREATE TABLE statement. These temporary tables are Sybase IQ-declared temporary tables and are available only in the current connection. For information about declared temporary tables in Sybase IQ, see the "DECLARE LOCAL TEMPORARY TABLE statement" in *Reference: Statements and Options*.

See the "CREATE TABLE statement" in *Reference: Statements and Options*.

Locating tables

Physical placement of a table is carried out differently in Adaptive Server Enterprise and Sybase IQ. Sybase IQ supports the ON *segment-name* clause, but *segment-name* refers to a Sybase IQ dbspace.

CREATE DEFAULT, CREATE RULE, and CREATE DOMAIN statements

Sybase IQ provides an alternative means of incorporating rules:

- Adaptive Server Enterprise supports the Create Default and Create Rule statements to create named defaults.
- SQL Anywhere and Sybase IQ support the CREATE DOMAIN statement to achieve the same objective.

CREATE TRIGGER statement

Support for triggers differs as follows:

- SQL Anywhere supports both row-level and statement-level triggers.
- Adaptive Server Enterprise supports only statement-level triggers.

- Sybase IQ does not support triggers.

Note A trigger is effectively a stored procedure that is run automatically either immediately before or immediately after an INSERT, UPDATE, or DELETE as part of the same transaction, that can be used to cause a dependent change (for example, to automatically update the name of an employee's manager when the employee is moved to a different department). It can also be used to write an audit trail to identify which modifications made which changes to the database, and at what time.

CREATE INDEX statement

CREATE INDEX syntax differs slightly among the three products:

- Adaptive Server Enterprise and SQL Anywhere support clustered or nonclustered indexes, using the following syntax:

  ```
  CREATE [UNIQUE] [CLUSTERED] INDEX name
  ON table (column,...)
  ON dbspace
  ```

  Adaptive Server Enterprise also allows the NONCLUSTERED keyword, but for both products the default is NONCLUSTERED.

- Adaptive Server Enterprise CREATE INDEX statements work in SQL Anywhere because SQL Anywhere allows, but ignores, the keywords FILLFACTOR, IGNORE_DUP_KEY, SORTED_DATA, IGNORE_DUP_ROW, and ALLOW_DUP_ROW.

- SQL Anywhere CREATE INDEX syntax supports the VIRTUAL keyword for use by its Index Consultant, but not for actual query executions.

- Sybase IQ supports seven specialized index types: LF, HG, HNG, DATE, TIME, DTTM, and WD. Sybase IQ also supports a CMP index on the relationship between two columns of identical data type, precision, and scale. Sybase IQ defaults to creating an HG index unless the index type is specified in the CREATE INDEX statement:

  ```
  CREATE [UNIQUE] [type] INDEX name
  ```

```
ON table (column,...)
```

Note Sybase IQ also supports CREATE JOIN INDEX, which lets you create a prejoined index on a certain set of columns that are joined consistently and frequently in queries.

See Chapter 6, "Using Sybase IQ Indexes" in the *System Administration Guide: Volume 1*.

Users, groups, and permissions

There are some differences between the Adaptive Server Enterprise and SQL Anywhere and Sybase IQ models of users and groups.

How users connect

In Adaptive Server Enterprise, users connect to a server, and each user requires a login ID and password to the server as well as a user ID for each database they want to access on that server.

SQL Anywhere and Sybase IQ users do not require a server login ID. All SQL Anywhere and Sybase IQ users receive a user ID and password for a database.

User groups

All three products support user groups, so you can grant permissions to many users at one time. However, there are differences in the specifics of groups:

- Adaptive Server Enterprise allows each user to be a member of only one group.
- SQL Anywhere and Sybase IQ allow users to be members of multiple groups, and group hierarchies are allowed.

All three products have a public group, for defining default permissions. Every user automatically becomes a member of the public group.

Database object permissions

GRANT and REVOKE statements for granting permissions on individual database objects are very similar in all three products.

- All three products allow SELECT, INSERT, DELETE, UPDATE, and REFERENCES permissions on database tables and views, and UPDATE permissions on selected columns of database tables.

  For example, the following statement is valid in all three products:

  ```
  GRANT INSERT, DELETE
  ON TITLES
  ```

```
TO MARY, SALES
```

This statement grants permission to use the INSERT and DELETE statements on the TITLES table to user MARY and to the SALES group.

- All three products allow EXECUTE permissions to be granted on stored procedures.
- Adaptive Server Enterprise also supports GRANT and REVOKE on additional items:
  - Objects: columns within tables, columns within views, and stored procedures
  - User abilities: CREATE DATABASE, CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TABLE, CREATE VIEW
- SQL Anywhere and Sybase IQ require a user to have RESOURCE authority to create database objects. (A closely corresponding Adaptive Server Enterprise permission is GRANT ALL, used by a Database Owner.)
- All three products support the WITH GRANT OPTION clause, allowing the recipient of permissions to grant them in turn, although Sybase IQ does not permit WITH GRANT OPTION to be used on a GRANT EXECUTE statement.

Database-wide permissions

Adaptive Server Enterprise uses a different model for database-wide user permissions.

- SQL Anywhere and Sybase IQ employ DBA permissions to allow a user full authority within a database.
- The System Administrator in Adaptive Server Enterprise enjoys this permission for all databases on a server. However, DBA authority on a Sybase IQ database is different from the permissions of an Adaptive Server Enterprise Database Owner, who must use the Adaptive Server Enterprise SETUSER statement to gain permissions on objects owned by other users.

Adding users

Adaptive Server Enterprise requires a two-step process to add a user: sp_addlogin followed by sp_add_user.

SQL Anywhere and Sybase IQ add users in a single step.

Sybase IQ Login Management stored procedures, although not required to add or drop users, allow DBAs to add or drop Sybase IQ user accounts. When Sybase IQ User Administration is enabled, these Sybase IQ user accounts let DBAs control user connections and password expirations.

See Chapter 8, “Managing User IDs and Permissions” in *System Administration Guide: Volume 1* and Chapter 3, “Sybase IQ as a Data Server” in the *System Administration Guide: Volume 2*.

Although SQL Anywhere and Sybase IQ allow Adaptive Server Enterprise system procedures for managing users and groups, the exact syntax and function of these procedures differs in some cases. For more information, see Chapter 7, “System Procedures,” including “Adaptive Server Enterprise system procedures” on page 522.

Load formats

Load format support in the three products is as follows:

- Sybase IQ handles ASCII, BINARY, and BCP load formats.
- SQL Anywhere, in addition to ASCII and BINARY, also lets you import dBase, Excel, FoxPro, and Lotus file formats.
- Adaptive Server Enterprise handles ASCII and BINARY load formats through BCP.

Note The syntax of the Sybase IQ and SQL Anywhere LOAD statement is based on BCP and designed to offer exactly the same functionality.

Setting options for Transact-SQL compatibility

Set Sybase IQ database options using the SET OPTION statement. See “Transact-SQL compatibility options” in “Database Options,” *Reference: Statements and Options*, for a list of option settings required for compatible behavior.

Data manipulation language

This section provides some general guidelines for writing portable queries, then discusses specific query requirements.

General guidelines for writing portable SQL

When writing SQL for use on more than one database management system, make your SQL statements as explicit as possible. Even if more than one server supports a given SQL statement, it might be a mistake to assume that default behavior is the same on each system. General guidelines applicable to writing compatible SQL include:

- Spell out all of the available options, rather than using default behavior.
- Use parentheses to make the order of execution within statements explicit, rather than assuming identical default order of precedence for operators.
- Use the Transact-SQL convention of an @ sign preceding variable names for Adaptive Server Enterprise portability.
- Declare variables and cursors in procedures and batches immediately following a BEGIN statement. Sybase IQ requires this, although Adaptive Server Enterprise allows declarations to be made anywhere in a procedure or batch.
- Do not use reserved words from either Adaptive Server Enterprise or Sybase IQ as identifiers in your databases.

Writing compatible queries

There are two criteria for writing a query that runs on both Sybase IQ and Adaptive Server Enterprise databases:

- The data types, expressions, and search conditions in the query must be compatible.
- The syntax of the SELECT statement itself must be compatible.

Sybase IQ supports the following subset of the Transact-SQL SELECT statement.

Syntax

SELECT [**ALL** | **DISTINCT**] *select-list*
…[**INTO** *#temporary-table-name*]
…[**FROM** *table-spec*,
…   *table-spec*, …]
…[**WHERE** *search-condition*]
…[**GROUP BY** *column-name*, …]
…[**HAVING** *search-condition*]

…| [**ORDER BY** *expression* [**ASC** | **DESC**], …] |
| [**ORDER BY** *integer* [**ASC** | **DESC**], …] |

Parameters

select-list:
{ *table-name.** }…
{ * }…
{ *expression* }…
{ *alias-name = expression* }…
{ *expression as identifier* }…
{ *expression as T_string* }…

table-spec:
[*owner*.]*table-name*
… [[**AS**] *correlation-name*]
…

alias-name:
identifier | '*string*' | “*string*"

For a full description of the SELECT statement, see "SELECT statement" in *Reference: Statements and Options*.

The sections that follow provide details on several items to be aware of when writing compatible queries.

Subqueries

Sybase IQ currently provides support for subqueries that is somewhat different from that provided by Adaptive Server Enterprise and SQL Anywhere. Adaptive Server Enterprise and SQL Anywhere support subqueries in the ON clause; Sybase IQ does not currently support this.

UNION in subqueries is supported as follows:

- SQL Anywhere supports UNION in both correlated and uncorrelated subqueries.
- Sybase IQ supports UNION only in uncorrelated subqueries.
- Adaptive Server Enterprise does not support UNION in any subqueries.

SQL Anywhere supports subqueries in many additional places that a scalar value might appear in the grammar. Adaptive Server Enterprise and Sybase IQ follow the ANSI standard as to where subqueries can be specified.

GROUP BY clause

GROUP BY ALL support is as follows:

- Adaptive Server Enterprise supports GROUP BY ALL, which returns all possible groups including those eliminated by the WHERE clause and HAVING clause. These have the NULL value for all aggregates.
- SQL Anywhere does not support the GROUP BY ALL Transact-SQL extension.

ROLLUP and CUBE in the GROUP BY clause are supported as follows:

- Sybase IQ and SQL Anywhere support ROLLUP and CUBE in the GROUP BY clause.
- Adaptive Server Enterprise does not currently support ROLLUP and CUBE.

Adaptive Server Enterprise supports projecting nongrouped columns in the SELECT clause. This is known as extended group by semantics and returns a set of values. Sybase IQ supports and SQL Anywhere do not support extended group by semantics. Only SQL Anywhere supports the List() aggregate to return a list of values.

For information about using GROUP BY with OLAP functions, see Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2*.

COMPUTE clause

COMPUTE clause support is as follows:

- Adaptive Server Enterprise supports the Transact-SQL COMPUTE clause.
- SQL Anywhere and Sybase IQ do not support the Transact-SQL COMPUTE clause since it is not in the ANSI standard and this functionality is provided by most third-party front-end tools.

WHERE clause

The WHERE clause differs in support for the Contains() predicate, and treatment of trailing white space in the Like() predicate.

- Sybase IQ supports the Contains() predicate for word searches in character data (similar to Contains in MS SQL Server and Verity). Sybase IQ uses WORD indexes to optimize these, if possible.
- SQL Anywhere and Adaptive Server Enterprise do not support Contains().

Joins

This section discusses support for Transact-SQL outer joins and ANSI joins.

Transact-SQL outer joins

Supported syntax for outer joins can be summarized as follows:

- Adaptive Server Enterprise fully supports *= and =* Transact-SQL syntax for outer joins.
- SQL Anywhere and Sybase IQ support Transact-SQL outer joins, but reject some complex Transact-SQL outer joins that are potentially ambiguous.
- Sybase IQ does not support chained (nested) Transact-SQL outer joins. Use ANSI syntax for this type of multiple outer join.

Note Transact-SQL outer join syntax is deprecated in SQL Anywhere and Sybase IQ.

ANSI joins

Support for ANSI join syntax can be summarized as follows:

- Sybase IQ does not currently support subqueries in the ON clause.
- Adaptive Server Enterprise and SQL Anywhere support subqueries in the ON clause.

Full outer join support is as follows:

- SQL Anywhere and Sybase IQ support FULL OUTER JOIN.
- Adaptive Server Enterprise does not support FULL OUTER JOIN.

More information on outer joins

For detailed information on Transact-SQL outer joins, including ANSI syntax alternatives, see the white paper "Semantics and Compatibility of Transact-SQL Outer Joins," from MySybase at http://www.sybase.com/support/. Although written for SQL Anywhere, the information in this document also applies to Sybase IQ.

Null comparisons

Adaptive Server Enterprise has Transact-SQL extensions that permit predicates to compare the null value. For example, `{col} = Null` means `{col} Is Null`.

SQL Anywhere and Sybase IQ use ANSI semantics for null comparisons unless the ANSINULL option is set to OFF, in which case such comparisons are Adaptive Server Enterprise-compatible.

Note SQL Anywhere 8.0 and later adds support for the TDS_EMPTY_STRING_AS_NULL to offer Adaptive Server Enterprise compatibility in mapping empty strings to the null value.

Zero-length strings

Zero-length strings are treated as follows:

- Adaptive Server Enterprise treats zero-length strings as the null value.

  Adaptive Server Enterprise users store a single space for blank strings.

- SQL Anywhere and Sybase IQ follow ANSI semantics for zero-length strings, that is, a zero-length string is a real value; it is not null.

HOLDLOCK, SHARED, and FOR BROWSE

Support for this syntax is as follows:

- Adaptive Server Enterprise supports HOLDLOCK, SHARED, and FOR BROWSE syntax.

- SQL Anywhere supports HOLDLOCK but does not support SHARED or FOR BROWSE.
- Sybase IQ does not support these keywords.

SQL functions

Sybase IQ supports most of the same functions as SQL Anywhere and Adaptive Server Enterprise, with these differences:

- Adaptive Server Enterprise supports the USING CHARACTERS | USING BYTES syntax in PatIndex(); SQL Anywhere and Sybase IQ do not.
- Adaptive Server Enterprise supports the Reverse() function; SQL Anywhere and Sybase IQ do not.
- Adaptive Server Enterprise supports Len() as alternative syntax for Length(); SQL Anywhere does not support this alternative.
- Adaptive Server Enterprise supports the Square() and Str_Replace() Microsoft compatibility functions; SQL Anywhere does not.
- Sybase IQ supports Str_Replace().
- Adaptive Server Enterprise and SQL Anywhere support TSEQUAL() to compare two timestamps for modification time; Sybase IQ does not support TSEQUAL(). (TSEQUAL is not relevant in the Sybase IQ table-level versioning model.)
- Sybase IQ supports ROWID(); Adaptive Server Enterprise and SQL Anywhere do not.
- SQL Anywhere and Sybase IQ support Cast() in addition to Adaptive Server Enterprise's Convert() for data type conversions.

Note Cast() is the ANSI-compliant name.

- SQL Anywhere and Sybase IQ support Lcase() and Ucase() as synonyms of Lower() and Upper(); Adaptive Server Enterprise does not.
- SQL Anywhere and Sybase IQ support the Locate() string function; Adaptive Server Enterprise does not.

- SQL Anywhere supports the IsDate() and IsNumeric() function to test the ability to convert a string to the respective data type; Adaptive Server Enterprise does not. Sybase IQ supports IsDate(). You can use IsNumeric in Sybase IQ, but CIS functional compensation performance considerations apply.
- SQL Anywhere supports the NEWID, STRTOUID, and UUIDTOSTR functions; Adaptive Server Enterprise does not. These are native functions in Sybase IQ, so CIS functional compensation performance considerations do not apply.

Note Some SQL functions, including SOUNDEX and DIFFERENCE string functions, and some date functions operate differently in Sybase IQ and SQL Anywhere. The Sybase IQ database option ASE_FUNCTION_BEHAVIOR specifies that output of some of the Sybase IQ data type conversion functions, including HEXTOINT and INTTOHEX, is consistent with the output of Adaptive Server Enterprise functions.

OLAP functions

Sybase IQ currently supports these OLAP functions:

- Corr()
- Covar_Pop()
- Covar_Samp()
- Cume_Dist
- Dense_Rank()
- Exp_Weighted_Avg
- First_Value
- Last_Value
- Median
- Ntile()
- Percent_Rank()
- Percentile_Cont()
- Percentile_Disc()

- Rank()
- Regr_Avgx()
- Regr_Avgy()
- Regr_Intercept()
- Regr_R2
- Regr_Slope()
- Regr_Sxx()
- Regr_Sxy()
- Regr_Syy()
- StdDev()
- Stddev_Pop
- Stddev_Samp
- Var_Pop
- Var_Samp
- Variance()
- Weighted_Avg

SQL Anywhere supports all of the Sybase IQ OLAP functions.

Currently, Adaptive Server Enterprise does not support OLAP functions.

CIS functional compensation does not support OLAP functions.

Note Support for OLAP functions is a rapidly evolving area of Sybase product development. See Chapter 4, "SQL Functions." Also see Chapter 2, "Using OLAP" in the *System Administration Guide: Volume 2.*

System functions

SQL Anywhere and Sybase IQ do not support the following Adaptive Server Enterprise system functions:

- curunreservedpgs() – number of pages free on a dbspace.
- data_pgs() – number of pages used by a table or index.

- host_id() – UNIX pid of the server process.
- hos_name() – name of the machine on which the server is running.
- lct_admin() – manages the “last chance threshold” for the Transaction manager.
- reserved_pgs() – number of pages allocated to a table or index.
- rowcnt() – number of rows in the specified table.
- valid_name() – whether a name would be a valid name if used, for example, for a table.
- valid_user() – returns TRUE if that user has connect permissions.
- ptn_data_pgs() – number of data pages in a partition.
- index_colorder() – returns the column order in an index.

User-defined functions

User-defined function (UDF) support varies as follows:

- SQL Anywhere supports UDFs in SQL, Java, and C.
- Adaptive Server Enterprise supports UDFs written only in Java .
- Sybase IQ offers support for UDFs via CIS query decomposition, but there are performance implications.

Arithmetic expressions on dates

SQL Anywhere and Sybase IQ interpret arithmetic expressions on dates as shorthand notation for various date functions. Adaptive Server Enterprise does not.

- Date +/- integer is equivalent to Dateadd().
- Date – date is equivalent to Datediff().
- Date + time creates a timestamp from the two.

SELECT INTO

There are differences in the types of tables permitted in a statement like the following:

```
select into table1 from table2
```

- Adaptive Server Enterprise permits *table1* to be permanent, temporary or a proxy table. Adaptive Server Enterprise also supports SELECT INTO EXISTING TABLE.
- SQL Anywhere and Sybase IQ permit *table1* to be a permanent or a temporary table. A permanent table is created only when you select into *table* and specify more than one column. SELECT INTO *#table*, without an owner specification, always creates a temporary table, regardless of the number of columns specified. SELECT INTO table with just one column selects into a host variable.

Updatable views

Adaptive Server Enterprise and SQL Anywhere are more liberal than ANSI permits on the view definitions that are updatable when the WITH CHECK option is not requested.

SQL Anywhere offers the ANSI_UPDATE_CONSTRAINTS option to control whether updates are restricted to those supported by SQL92, or a more liberal set of rules.

Sybase IQ permits UPDATE only on single-table views that can be flattened. Sybase IQ does not support WITH CHECK.

FROM clause in UPDATE and DELETE

All three products support the FROM clause with multiple tables in UPDATE and DELETE.

Transact-SQL procedure language overview

The **stored procedure language** is the part of SQL used in stored procedures and batches.

SQL Anywhere and Sybase IQ support a large part of the Transact-SQL stored procedure language in addition to the Watcom-SQL dialect based on SQL92.

Transact-SQL stored procedure overview

Because it is based on the ISO/ANSI draft standard, the SQL Anywhere and Sybase IQ stored procedure language differs from the Transact-SQL dialect in many ways. Many of the concepts and features are similar, but the syntax is different. SQL Anywhere and Sybase IQ support for Transact-SQL takes advantage of the similar concepts by providing automatic translation between dialects. However, you must write a procedure exclusively in one of the two dialects, not in a mixture of the two.

There are a variety of aspects to SQL Anywhere and Sybase IQ support for Transact-SQL stored procedures, including:

- Passing parameters
- Returning result sets
- Returning status information
- Providing default values for parameters
- Control statements
- Error handling

Transact-SQL batch overview

In Transact-SQL, a batch is a set of SQL statements submitted together and executed as a group, one after the other. Batches can be stored in command files. The ISQL utility in SQL Anywhere and Sybase IQ and the *isql* utility in Adaptive Server Enterprise provide similar capabilities for executing batches interactively.

The control statements used in procedures can also be used in batches. SQL Anywhere and Sybase IQ support the use of control statements in batches and the Transact-SQL-like use of nondelimited groups of statements terminated with a GO statement to signify the end of a batch.

For batches stored in command files, SQL Anywhere and Sybase IQ support the use of parameters in command files. Adaptive Server Enterprise does not support parameters.

For information on parameters, see "PARAMETERS statement [DBISQL]" in *Reference: Statements and Options*.

SQL statements in procedures and batches

Some SQL statements supported by Sybase IQ are part of one dialect, but not the other. You cannot mix the two dialects within a procedure or batch. This means that:

- You can include Transact-SQL-only statements with statements that are part of both dialects in a batch or procedure.
- You can include statements not supported by Adaptive Server Enterprise with statements that are supported by both servers in a batch or procedure.
- You cannot include Transact-SQL–only statements with Sybase IQ—only statements in a batch or procedure.

SQL statements *not* separated by semicolons are part of a Transact-SQL procedure or batch. See *Reference: Statements and Options* for details of individual statements.

Expression subqueries in IF statements

Adaptive Server Enterprise and SQL Anywhere support comparisons between a variable and a scalar value returned by an expression subquery. For example:

```
create procedure testIf ()
   begin
   declare var4 int;
set var4 = 10;
   if var4 = (select MIN (a_i1) from a) then set
   var4 = 100;
end if;
end;
```

CASE statement

Permitted usage of the CASE statement differs in Sybase IQ and SQL Anywhere.

The CASE statement is not supported in Adaptive Server Enterprise, which supports case expressions only.

For a detailed comparison of case expression support in Sybase IQ and Adaptive Server Enterprise, see “Expressions” on page 23.

Row-level cursor operations

All three products support the use of cursors with UPDATE and DELETE as follows:

```
UPDATE WHERE CURRENT OF {cursor}
```

```
DELETE WHERE CURRENT OF {cursor}
```

In Sybase IQ, updatable cursors are asensitive only, for one table only, and chained only. Updatable hold cursors are not permitted. Updatable cursors in Sybase IQ get a table lock.

Print command

The effect of the PRINT command depends on the client:

- Adaptive Server Enterprise PRINT always sends a message to the client.
- In SQL Anywhere and Sybase IQ, PRINT sends a message to the client for Open Client and JDBC connections.
- Adaptive Server Enterprise stored procedures that rely on PRINT work in Sybase IQ using DBISQL.

Note Sybase IQ users might prefer dbisql with JDBC, rather than the iAdaptive Server Anywhere JDBC driver (formerly called the JDBC-ODBC bridge).

Automatic translation of stored procedures

In addition to supporting Transact-SQL alternative syntax, SQL Anywhere and Sybase IQ provide aids for translating statements between the Watcom-SQL and Transact-SQL dialects. Functions returning information about SQL statements and enabling automatic translation of SQL statements include:

- **SQLDialect(statement)** Returns Watcom-SQL or Transact-SQL.
- **WatcomSQL(statement)** Returns the Watcom-SQL syntax for the statement.
- **TransactSQL(statement)** Returns the Transact-SQL syntax for the statement.

These are functions and thus can be accessed using a SELECT statement from ISQL. For example, the following statement returns the value Watcom-SQL:

```
SELECT SqlDialect('select * from Employees')
```

Returning result sets from Transact-SQL procedures

SQL Anywhere and Sybase IQ use a RESULT clause to specify returned result sets. In Transact-SQL procedures, column names or alias names of the first query are returned to the calling environment.

Example of Transact-SQL procedure

The following Transact-SQL procedure illustrates how Transact-SQL stored procedures return result sets:

```
CREATE PROCEDURE showdept (@deptname varchar(30))
AS
    SELECT Employees.Surname, Employees.GivenName
    FROM Departments, Employees
    WHERE Departments.DepartmentName = @deptname
    AND Departments.DepartmentID =
Employees.DepartmentID
```

Example of Watcom-SQL procedure

The following is the corresponding SQL Anywhere or Sybase IQ procedure:

```
CREATE PROCEDURE showdept(in deptname varchar(30))
RESULT ( lastname char(20), firstname char(20))
BEGIN
```

```
    SELECT Employees.Surname, Employees.GivenName
    FROM Departments, Employees
    WHERE Departments.DepartmentName = deptname
    AND Departments.DepartmentID =
Employee.DepartmentID
END
```

Multiple result sets

There are minor differences in the way the three Sybase client tools present multiple results to the client:

- ISQL displays all results in a single stream.
- DBISQL presents each result set on a separate tab. You must enable this functionality in the Option menu. Make it a permanent change, then restart or reconnect to DBISQL.
- DBISQLC provides the RESUME command to display each successive result set.

For more information about procedures and results, see Chapter 1, "Using Procedures and Batches" in the *System Administration Guide: Volume 2*.

Variables in Transact-SQL procedures

SQL Anywhere and Sybase IQ use the SET statement to assign values to variables in a procedure. In Transact-SQL, values are assigned using the SELECT statement with an empty table list. The following simple procedure illustrates how the Transact-SQL syntax works:

```
CREATE PROCEDURE multiply
                @mult1 int,
                @mult2 int,
                @result int output
AS
SELECT @result = @mult1 * @mult2
```

This procedure can be called as follows:

```
CREATE VARIABLE @product int
go
EXECUTE multiply 5, 6, @product OUTPUT
go
```

The variable *@product* has a value of 30 after the procedure executes.

Order and persistence of variables

There are some differences in order and persistence of variable declarations:

- In Adaptive Server Enterprise, you can declare variables anywhere in the body of a stored procedure. Variables persist for the duration of the procedure.
- In SQL Anywhere and Sybase IQ, you must declare variables at the beginning of a compound statement (that is, immediately after BEGIN in a BEGIN...END pair). Variables persist only for the duration of the compound statement.

For more information on using the SELECT statement to assign variables, see “Writing compatible queries” on page 617. For more information on using the SET statement to assign variables, see "SET statement [ESQL]" in *Reference: Statements and Options*.

Error handling in Transact-SQL procedures

Default procedure error handling is different in the Watcom-SQL and Transact-SQL dialects. By default, Watcom-SQL dialect procedures exit when they encounter an error, returning SQLSTATE and SQLCODE values to the calling environment.

You can build explicit error handling into Watcom-SQL stored procedures using the EXCEPTION statement, or you can instruct the procedure to continue execution at the next statement when it encounters an error, using the ON EXCEPTION RESUME statement.

When a Transact-SQL dialect procedure encounters an error, execution continues at the following statement. The global variable @@error holds the error status of the most recently executed statement. You can check this variable following a statement to force return from a procedure. For example, the following statement causes an exit if an error occurs:

```
IF @@error != 0 RETURN
```

When the procedure completes execution, a return value indicates the success or failure of the procedure. This return status is an integer, and can be accessed as follows:

```
DECLARE @status INT
EXECUTE @status = proc_sample
IF @status = 0
```

```
    PRINT 'procedure succeeded'
ELSE
    PRINT 'procedure failed'
```

Table A-2 describes the built-in procedure return values and their meanings:

Table A-2: Built-in procedure return values

Value	Meaning
0	Procedure executed without error
-1	Missing object
-2	Data type error
-3	Process was chosen as deadlock victim
-4	Permission error
-5	Syntax error
-6	Miscellaneous user error
-7	Resource error, such as out of space
-8	Nonfatal internal problem
-9	System limit was reached
-10	Fatal internal inconsistency
-11	Fatal internal inconsistency
-12	Table or index is corrupt
-13	Database is corrupt
-14	Hardware error

The RETURN statement can be used to return other integers, with their own user-defined meanings.

Using the RAISERROR statement in procedures

The RAISERROR statement is a Transact-SQL statement for generating user-defined errors. It has a similar function to the SIGNAL statement.

For a description of the RAISERROR statement, see "RAISERROR statement [T-SQL]" in *Reference: Statements and Options*.

By itself, RAISERROR does not cause an exit from the procedure, but it can be combined with a RETURN statement or a test of the @@error global variable to control execution following a user-defined error.

If you set the ON_TSQL_ERROR database option to CONTINUE, RAISERROR no longer signals an execution-ending error. Instead, the procedure completes and stores the RAISERROR status code and message, and returns the most recent RAISERROR. If the procedure causing the RAISERROR was called from another procedure, RAISERROR returns after the outermost calling procedure terminates.

You lose intermediate RAISERROR statuses and codes when the procedure terminates. If, at return time, an error occurs along with RAISERROR, the error information is returned and you lose the RAISERROR information. The application can query intermediate RAISERROR statuses by examining @@error global variable at different execution points.

Transact-SQL-like error handling in the Watcom-SQL dialect

You can make a Watcom-SQL dialect procedure handle errors in a Transact-SQL-like manner by supplying the ON EXCEPTION RESUME clause to the CREATE PROCEDURE statement:

```
CREATE PROCEDURE sample_proc()
ON EXCEPTION RESUME
BEGIN
    ...
END
```

The presence of an ON EXCEPTION RESUME clause prevents explicit exception handling code from being executed, so avoid using these two clauses together.

SQL Anywhere and Sybase IQ

The preceding sections, while focused on compatibility with Transact-SQL, also clarify many of the distinctions between Sybase IQ and SQL Anywhere.

This section points out other differences and shared functionality between Sybase IQ and SQL Anywhere.

For additional information, always refer to the Sybase IQ documentation set when using the product. Refer to the SQL Anywhere documentation set when using SQL Anywhere, or when the Sybase IQ documentation refers to SQL Anywhere documentation for specific functionality only.

Server and database startup and administration

Note the following differences in starting and managing databases and servers:

- Sybase IQ uses the server startup command start_iq, instead of the SQL Anywhere network server startup command.
- Sybase IQ does not support personal servers.
- Sybase IQ supports many SQL Anywhere server command line options, but not all. Other server options are supported for Sybase IQ but not for SQL Anywhere.
- Sybase IQ provides the stop_iq utility to shut down servers.
- Clauses permitted in the BACKUP and RESTORE statements differ in Sybase IQ and SQL Anywhere.
- SQL Remote is supported in Sybase IQ only for multiplex operations.

Sybase IQ supports many SQL Anywhere database administration utilities, but not all:

- The following SQL Anywhere utilities are *not* supported by Sybase IQ: backup, compression, console, initialization, license, log transfer, log translation, rebuild, spawn, some transaction log options (-g, -il, -ir, -n, -x, -z), uncompression, unload, upgrade, and write file.
- Sybase IQ supports the SQL Anywhere validation utility only on the catalog store. To validate the IQ store, use sp_iqcheckdb.

Database options

Some SQL Anywhere database options are not supported by Sybase IQ, including DEFAULT_TIMESTAMP_INCREMENT.

Some database options apply only to the catalog store, including: FOR_XML_NULL_TREATMENT, ISOLATION_LEVEL, PREFETCH, PRECISION, and SCALE.

Options with differences in behavior, default, or allowed values include DELAYED_COMMITS, TIME_FORMAT, TIMESTAMP_FORMAT.

Sybase IQ also includes many options that SQL Anywhere does not support. See Chapter 2, "Database Options." in *Reference: Statements and Options*.

Data definition language (DDL)

In addition to the DDL differences discussed previously:

- In a DELETE/DROP or PRIMARY KEY clause of an ALTER TABLE statement, Sybase IQ takes the RESTRICT action (reports an error if there are associated foreign keys). SQL Anywhere always takes the CASCADE action.
- Similarly, DROP TABLE statement reports an error in Sybase IQ if there are associated foreign-key constraints.
- Sybase IQ does not support these DDL statements: CREATE COMPRESSED DATABASE, CREATE TRIGGER, SETUSER.
- Sybase IQ supports referential integrity at the statement level, rather than the transaction-level integrity that SQL Anywhere supports with the CHECK ON COMMIT clause of the CREATE TABLE statement.
- A Sybase IQ table cannot have a foreign key that references a SQL Anywhere (or catalog) table, and a SQL Anywhere table cannot have a foreign key that references a Sybase IQ table.
- In a Sybase IQ database, publications can only be created on SQL Anywhere tables.
- In CREATE DATABASE, the defaults for case-sensitivity and collation differ. The defaults for Sybase IQ are CASE RESPECT and the ISO_BINENG collation; for SQL Anywhere, the defaults are CASE IGNORE, and collation inferred from the language and character set of the operating system.

Data manipulation language (DML)

- Sybase IQ does not support these DML and procedural statements: EXPLAIN, GET DATA, INPUT, PREPARE TO COMMIT, PUT, READTEXT, ROLLBACK TRIGGER, SYSTEM, UNLOAD TABLE, VALIDATE TABLE.

  Note A set of extraction options perform a role similar to UNLOAD TABLE; for details, see the section “Data extraction options” in Chapter 7, “Moving Data In and Out of Databases” in the *System Administration Guide: Volume 1*.

- Sybase IQ supports the INSERT...LOCATION syntax; SQL Anywhere does not.
- LOAD TABLE options differ in Sybase IQ and SQL Anywhere.
- OPEN statement in Sybase IQ does not support BLOCK and ISOLATION LEVEL clauses.
- Sybase IQ does not support triggers.
- Use of transactions, isolation levels, checkpoints, and automatically generated COMMITs, as well as cursor support, is different in Sybase IQ and SQL Anywhere. See Chapter 10, “Transactions and Versioning” in the *System Administration Guide: Volume 1*.
- When you SELECT from a stored procedure in Sybase IQ, CIS functional compensation performance considerations apply. See “Conditions that cause processing by SQL Anywhere” in *Performance and Tuning Guide*.
- Sybase IQ ignores the database name qualifier in fully qualified names in Adaptive Server Enterprise SELECT statements, such as a FROM clause with ***<database name>.<owner>.<table name>***. For example, Sybase IQ interprets the query `SELECT * FROM XXX..TEST` as `SELECT * FROM TEST`.

Stored procedures

See “SQL Anywhere supported procedures” on page 525.

Obsolete procedures and views are removed from databases created in versions earlier than Sybase IQ 12.6 ESD #2 when you run the ALTER DATABASE UPGRADE command using Sybase IQ 12.6 ESD #2 or later.

If you do not run ALTER DATABASE UPGRADE, the procedures remain in the database.

Adaptive Server Enterprise and Sybase IQ

This section points out other differences and shared functionality between Sybase IQ and Adaptive Server Enterprise.

For additional information, always refer to the Sybase IQ documentation set when using the product. Refer to the Adaptive Server Enterprise documentation set when using Adaptive Server Enterprise, or when the Sybase IQ documentation refers to Adaptive Server Enterprise documentation for specific functionality only.

Stored procedures

Sybase IQ no longer supports these Adaptive Server stored procedures:

- sp_addserver
- sp_configure
- sp_estspace
- sp_help
- sp_helpuser
- sp_who

Sybase IQ no longer supports the following catalog procedures:

- sp_column_privileges
- sp_databases
- sp_datatype_info
- sp_server_info

Obsolete procedures and views are removed from databases created prior to Sybase IQ 15.1 ESD #2 when you run the ALTER DATABASE UPGRADE command using Sybase IQ 15.1 ESD #2 or higher.

If you do not run ALTER DATABASE UPGRADE, the procedures remain in the database.

System views

Sybase IQ no longer supports these Adaptive Server Enterprise views:

- sysalternates
- sysaudits
- sysauditoptions
- sysconstraints
- syscharsets
- sysconfigures
- syscurconfigs
- sysdatabases
- sysdepends
- sysdevices
- sysengines
- syskeys
- syslanguages
- syslocks
- syslogs
- sysloginroles
- sysmessages
- sysprocedures
- sysprocesses
- sysprotects
- sysreferences
- sysremotelogins
- sysroles

- syssegments
- sysservers
- syssrvroles
- systhresholds
- sysusages

Column name differences

The column name used in the Adaptive Server Enterprise view SYSTYPES is “allownulls”. The column name used in the Sybase IQ view SYSTYPES is “allowsnulls”.

Index

C

D

E

F

G

H

I

J

N

O

P

Q

R

S

T

X

Y

Z