SYBASE®

Reference Manual

Sybase® IQ

12.7

DOCUMENT ID: DC38151-01-1270-01

LAST REVISED: June 2006

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 start-up parameters, see the *Sybase IQ Utility Guide*.

Audience

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

How to use this book

This book provides comprehensive descriptions of Sybase IQ features, but it does not describe why you might want to use each feature. This book is designed to be used as a reference together with the other books in the Sybase IQ documentation set.

Windows platforms

The Windows information in this book applies to all supported Windows platforms, unless otherwise noted. For supported Windows platforms, see the *Release Bulletin* Sybase IQ *for Windows.*

Related documents

The Sybase IQ document set consists of these documents:

- *Introduction to Sybase IQ* – contains information and exercises for users unfamiliar with Sybase IQ and with the Sybase Central™ database management tool.
- *New Features in Sybase IQ 12.7* – includes a brief description of new features in Sybase IQ.
- *Sybase IQ Performance and Tuning Guide* – describes query optimization, design, and tuning issues for very large databases.
- *Sybase IQ System Administration Guide* – describes administrative concepts, procedures and performance tuning recommendations supported by Sybase IQ, including how to manage the IQ Store.
- *Sybase IQ Troubleshooting and Recovery Guide* – Shows how to solve problems and perform system recovery and database repair.

- *Sybase IQ Error Messages* – refers to IQ error messages which are referenced by SQLCode, SQLState, and Sybase error code, and SQL preprocessor errors and warnings.
- *Sybase IQ Utility Guide* – contains Sybase IQ utility program reference material, such as available syntax, parameters, and options.
- *Large Objects Management in Sybase IQ* – describes 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.
- *Sybase IQ Installation and Configuration Guide* – contains platform-specific instructions on installing Sybase IQ, migrating to a new version of Sybase IQ, and configuring Sybase IQ for a particular platform.
- *Sybase IQ Release Bulletin* – contains last-minute changes to the product and documentation.
- *Encrypted Columns in Sybase IQ* – describes the use of user encrypted columns within the Sybase IQ data repository. You need a separate license to install this product option.

Sybase IQ and Adaptive Server Anywhere

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

Documentation for Adaptive Server Anywhere:

- *Adaptive Server Anywhere Programming Guide* – Intended for application developers writing programs that directly access the ODBC, Embedded SQL™, or Open Client™ interfaces, this book describes how to develop applications for Adaptive Server Anywhere.
- *Adaptive Server Anywhere Database Administration Guide* – Intended for all users, this book covers material related to running, managing, and configuring databases and database servers.
- *Adaptive Server Anywhere SQL Reference Manual* – Intended for all users, this book provides a complete reference for the SQL language used by Adaptive Server Anywhere. It also describes the Adaptive Server Anywhere system tables and procedures.

You can also refer to the Adaptive Server Anywhere documentation in the SQL Anywhere Studio 9.0.2 collection on the Sybase Product Manuals Web site. To access this site, go to Product Manuals at http://www.sybase.com/support/manuals/.

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 might 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://www.sybase.com/support/manuals/.

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://www.sybase.com/support/techdocs/.
2. Click Certification Report.
3. In the Certification Report filter select a product, platform, and timeframe and then click Go.

4 Click a Certification Report title to display the report.

❖ **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 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, type your MySybase user name and password.

3 Select a product.

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

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 the following syntax 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 is the same as Select, which is the same as select.
- **Placeholders** Items that must be replaced with appropriate identifiers or expressions are shown in *italics*.
- **Continuation** Lines beginning with an ellipses (...) 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 more than one is 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 ]
  ```

  It indicates that the *savepoint-name* is optional. Do not type the square 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 ]
  ```

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

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

  You must include either ON or OFF Do not type the braces.

Typographic conventions

Table 1 lists the typographic conventions that this documentation uses.

Table 1: Typographic conventions

Item	Description
`Code`	SQL and program code is displayed in a mono-spaced (fixed-width) font.
`User entry`	Text entered by the user is shown in serif type.
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 san-serif type in print, and in italic online.

The sample database

Sybase IQ includes a sample database used by many of the examples in the Sybase IQ documentation.

The sample database represents a small company. It contains internal information about the company (employees, departments, and financial data), as well as product information (products), sales information (sales orders, customers, and contacts), and financial information (fin_code, fin_data).

The sample database is held in a file named *asiqdemo.db*, located in the directory *$ASDIR/demo* on UNIX systems and *%ASDIR%\demo* on Windows systems.

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 12.7 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.

For information about accessibility support in the Sybase IQ plug-in for Sybase Central, see “Using accessibility features” in *Introduction to Sybase IQ*. The online help for this product, which you can navigate using a screen reader, also describes accessibility features, including Sybase Central keyboard shortcuts.

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 and see “Using screen readers” in *Introduction to Sybase IQ*.

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 are chosen 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 *ASIQ-12_7*. The location of *ASIQ-12_7* varies, depending on where you install Sybase IQ. The *ASIQ-12_7* directory is also referred to by the environment variable *$ASDIR* on UNIX or *%ASDIR%* on Windows.

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

- *Demo directory* (*ASIQ-12_7/demo*) – holds the sample database used in documentation examples. The database is held in the files *asiqdemo.db*, *asiqdemo.iq*, *asiqdemo.iqmsg*, and *asiqdemo.iqtmp*. When you start the database, an *asiqdemo.log* file is also created. This directory is not essential, but Sybase recommends that you keep it.

  The subdirectory */demo/demodata* holds the data to create the demo database, *asiqdemo*. You can use *demo/mkasiqdemo.sql* to re-create the demo database. The sample database can be used to demonstrate problems to Technical Support.

- *Scripts directory* (*ASIQ-12_7/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* (*/asa/c* and */asa/javaSQL*) – */c* holds C++ examples that illustrate using ESQL (embedded SQL) and C with Adaptive Server Anywhere. Because Adaptive Server Anywhere and Sybase IQ share common code, you can modify these examples for use with Sybase IQ. The */javaSQL* directory holds Java examples.

- *Executable directories* – hold executables, libraries, help files, and the like.

On UNIX, executable subdirectories include ASIQ-12_7 subdirectories */bin*, */lib*, */logfiles*, */res*, */tix*, and */usrlib*. On Windows, these include ASIQ-12_7 subdirectories *\h*, *\install*, *\java*, and *\win32*.

- *Readme file* – contains the latest information about installing and running Sybase IQ. Sybase strongly suggests that you print this file and read it.

  ASIQ-12_7/readme.txt on UNIX or *ASIQ-12_7\readme.txt* on Windows

How Sybase IQ locates files

When starting and running, 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 Adaptive Server Anywhere libraries. (If either of these products have already been installed on your system, you should know the directory where they are installed, to avoid confusion.)

The types of files include but are not limited to:

- *Libraries* – might include product libraries, system libraries, or Adaptive Server Enterprise libraries. File name extensions include *.so.nnn* or *.so* on UNIX (*.sl.nnn* or *.sl* on HP), or *.dll* or *.lib* on Windows. These files are required to run Sybase IQ. If an incorrect DLL is located, for example, there is the possibility of version mismatch errors. For example, library files might be found in *$ASDIR/lib* or *$SYBASE/$SYBASE_OCS/lib* on UNIX, or *%ASDIR%\win32* or *%SYBASE\%SYBASE_OCS\dll* on Windows. An empty directory, *$ASDIR/usrlib*, lets you supersede default libraries with custom libraries and patches, because start_asiq 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 Chapter 4, “Configuring Sybase IQ” in the *Sybase IQ Installation and Configuration Guide*.
- *Configuration files* – used to specify connection parameters. Examples include *default.cfg* on Windows or *asiqdemo.cfg*.
- *Database files* – store the data and metadata. For example: *asiqdemo.db*, *asiqdemo.iq*, *asiqdemo.iqmsg*, *asiqdemo.iqtmp*.

- *Log files* – store information about the current session on the server and connected database. For example, a server log might be named *ASIQ-12_7/logfiles/janed_asiqdemo.006.srvlog*. The database log (for example, *ASIQ-12_7/demo/asiqdemo.log*) is created when you connect to the database and stored in the directory where the server is started. For more information about these files, see the *Sybase IQ 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*. The UNIX server does not search in this location.

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 (*win32*, *win*, 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 *LD_LIBRARY_PATH_64*, *LIBPATH*, or *SHLIB_PATH* (depending on platform) 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. These environment variables are listed in this section.

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/ASIQ-12_7/ASIQ-12_7.sh
```

2 For the C shell:

```
source $SYBASE/ASIQ-12_7/ASIQ-12_7.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/janed/sybase
```

2 In one of your start-up 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 the following topics:

- "SYBASE_OCS environment variable" on page 14
- "TZ environment variable" on page 14

ASCHARSET environment variable

Setting **ASCHARSET**=*charset*

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

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

- ASCHARSET environment variable
- Query the operating system

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

ASDIR environment variable

Setting `ASDIR = ${SYBASE}/ASIQ-12_7`

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

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

- *$ASDIR/bin/util_db.ini* holds the login ID and password for the utility database, utility_db. The installation program lets you change these from their default values, login ID "dba" and password "sql."
- *$ASDIR/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 ASLOGDIR environment variable.
- *$ASDIR/demo* is the location for the asiqdemo database files.

ASIQPORT environment variable

Setting

```
ASIQPORT = 5556
```

Operating system

Optional. If the user did not specify ASIQPORT 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 -DASIQPORT argument to the scjview command when starting Sybase Central. For example:

```
scjview -DASIQPORT=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 plug-in 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.

This functionality lets you run IQ Agents for Sybase IQ 12.6 and 12.7 on the same system. It also lets you run any number of 12.7 IQ Agents on a given host.

ASIQTIMEOUT environment variable

Setting

ASIQTIMEOUT = *nnn*

Operating system

Optional but recommended in multiplex environments.

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.

This variable 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):

```
ASIQTIMEOUT=45
export ASIQTIMEOUT
```

- To wait an hour (C shell):

```
setenv ASIQTIMEOUT 60
```

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

ASLANG environment variable

Setting: **ASLANG**=*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 ASLANG=DE sets the default language to German.

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

- ASLANG 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.

ASLOGDIR environment variable

Setting: `ASLOGDIR = path`

Operating system: Optional.

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

- The backup log is *.backup.syb*, in the directory specified by $ASLOGDIR.
- The server log is in the file *servername.nnn.srvlog* (where nnn is the number of times the server has been started) in the directory specified by $ASLOGDIR.

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

ASTMP environment variable

Setting

```
ASTMP = temp_directory
```

Operating system

Optional on UNIX. Not used on Windows platforms.

Description

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

The ASTMP environment variable should point to a local directory for those using NFS (network file system), which permits the ASTMP 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 ASTMP 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 on page 442.

If you do not set ASTMP explicitly, or if it is set to $SYBASE or $ASDIR, then the Sybase IQ Agent sets ASTMP 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. (Sybase IQ uses shared memory connectivity instead of network connectivity when you do not specify the port or engine number for connection.)

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 ASTMP 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. For details, see the *Sybase IQ 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
```

LIBRARY PATH environment variable

Settings

```
For AIX:
LIBPATH = installation_path/lib

For HP UNIX:
SHLIB_PATH = installation_path/lib

For Solaris:
LD_LIBRARY_PATH_64 = installation_path/lib
```

Operating system

Required. Variable name is platform dependent. UNIX only.

Description

This variable is set to include 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. The environment source file adds the *$SYBASE/$SYBASE_OCS/bin* directory to your UNIX path.

On Windows, PATH takes the place of the LIBRARY_PATH variable, so executables and DLLs are located using the PATH variable. Installing Sybase IQ on Windows adds *%SYBASE%\%SYBASE_OCS%\bin* and *%SYBASE%\%SYBASE_OCS%\dll* to your Windows path.

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

For a description of the connection parameters, see "Connection parameters" in Chapter 4, "Connection and Communication Parameters" in the *Sybase IQ System Administration Guide*.

SYBASE environment variable

Setting

```
SYBASE = path
```

Operating system

Required.

Description

The SYBASE variable 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_JRE environment variable

Setting

```
SYBASE_JRE= "${SYBASE}/shared/jre-1_42"
```

Operating system

Required by SDK only.

Description

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

SYBASE_OCS environment variable

Setting

On Linux 32-bit systems:

```
SYBASE_OCS = "OCS-12_5
```

On all other systems:

```
SYBASE_OCS = "OCS-15_0"
```

Operating system

Required.

Description

On UNIX, run the environment source file to set SYBASE_OCS. Installing Sybase IQ on UNIX does not automatically reset SYBASE_OCS, and if the value has been set by another Sybase product, that value remains in effect unless you unset SYBASE_OCS, and then run the source file. See the *Sybase IQ Installation and Configuration Guide* for details.

On Windows, the installation program sets SYBASE_OCS when it installs Open Client/Server Software Developers Kit.

TZ environment variable

Setting

When using Component Integration Services (CIS) in certain geographic regions, connection attempts return the error No Suitable Driver. Java Developer Kits used with Sybase IQ 12.7 support only the time zone codes shown in Table 1-1 and Table 1-2.

- For databases using default JDK 1.1.8:

  Substitute JST for unsupported time zone KST, which gives the same GMT+9 time, as follows:

  ```
  setenv TZ JST
  ```

  See Table 1-1 for the appropriate time zone code and settings.

- For databases using JDK 1.3:

  ```
  setenv TZ Asia/Seoul
  ```

  See Table 1-2 for the appropriate time zone code and settings.

Description

Set the time zone environment variable to a supported setting, start the server, and CIS works as expected. To ensure that the correct setting is always used, you can set the time zone in the start_asiq script.

Table 1-1: JDK 1.1.8

Time zone setting	Time zone code	Time zone setting	Time zone code
0	GMT	16	SST
1	UTC	17	NST
2	ECT	18	MIT
3	EET	19	HST
4	ART	20	AST
5	EAT	21	PST
6	MET	22	PNT
7	NET	23	MST
8	PLT	24	CST
9	IST	25	EST
10	BST	26	IET
11	VST	27	PRT
12	CTT	28	CNT
13	JST	29	AGT
14	ACT	30	BET
15	AET	31	CAT

Table 1-2: JDK 1.3

Time zone setting	Time zone code	Time zone setting	Time zone code
0	Pacific/Niue	161	Europe/Belgrade
1	Pacific/Apia	162	Europe/Paris
2	MIT	163	ECT
3	Pacific/Pago_Pago	164	Africa/Bujumbura
4	Pacific/Tahiti	165	Africa/Gaborone
5	Pacific/Fakaofo	166	Africa/Lubumbashi
6	Pacific/Honolulu	167	Africa/Maseru
7	HST	168	Africa/Blantyre
8	America/Adak	169	Africa/Maputo
9	Pacific/Rarotonga	170	Africa/Kigali
10	Pacific/Marquesas	171	Africa/Khartoum
11	Pacific/Gambier	172	Africa/Mbabane
12	America/Anchorage	173	Africa/Lusaka
13	AST	174	Africa/Harare

Time zone setting	Time zone code	Time zone setting	Time zone code
14	Pacific/Pitcairn	175	CAT
15	America/Vancouver	176	Africa/Johannesburg
16	America/Tijuana	177	Europe/Sofia
17	America/Los_Angeles	178	Europe/Minsk
18	PST	179	Asia/Nicosia
19	America/Dawson_Creek	180	Europe/Tallinn
20	America/Phoenix	181	Africa/Cairo
21	PNT	182	ART
22	America/Edmonton	183	Europe/Helsinki
23	America/Mazatlan	184	Europe/Athens
24	America/Denver	185	Asia/Jerusalem
25	MST	186	Asia/Amman
26	America/Belize	187	Asia/Beirut
27	America/Regina	188	Europe/Vilnius
28	Pacific/Galapagos	189	Europe/Riga
29	America/Guatemala	190	Europe/Chisinau
30	America/Tegucigalpa	191	Europe/Bucharest
31	America/El_Salvador	192	Europe/Kaliningrad
32	America/Costa_Rica	193	Asia/Damascus
33	America/Winnipeg	194	Europe/Kiev
34	Pacific/Easter	195	Europe/Istanbul
35	America/Mexico_City	196	EET
36	America/Chicago	197	Asia/Bahrain
37	CST	198	Africa/Djibouti
38	America/Porto_Acre	199	Africa/Asmera
39	America/Bogota	200	Africa/Addis_Ababa
40	America/Guayaquil	201	EAT
41	America/Jamaica	202	Africa/Nairobi
42	America/Cayman	203	Indian/Comoro
43	America/Managua	204	Asia/Kuwait
44	America/Panama	205	Indian/Antananarivo
45	America/Lima	206	Asia/Qatar
46	America/Indianapolis	207	Africa/Mogadishu
47	IET	208	Africa/Dar_es_Salaam
48	America/Nassau	209	Africa/Kampala

Time zone setting	Time zone code	Time zone setting	Time zone code
49	America/Montreal	210	Asia/Aden
50	America/Havana	211	Indian/Mayotte
51	America/Port-au-Prince	212	Asia/Riyadh
52	America/Grand_Turk	213	Asia/Baghdad
53	America/New_York	214	Europe/Simferopol
54	EST	215	Europe/Moscow
55	America/Antigua	216	Asia/Tehran
56	America/Anguilla	217	MET
57	America/Curacao	218	Asia/Dubai
58	America/Aruba	219	Indian/Mauritius
59	America/Barbados	220	Asia/Muscat
60	America/La_Paz	221	Indian/Reunion
61	America/Manaus	222	Indian/Mahe
62	America/Dominica	223	Asia/Yerevan
63	America/Santo_Domingo	224	NET
64	America/Grenada	225	Asia/Baku
65	America/Guadeloupe	226	Asia/Aqtau
66	America/Guyana	227	Europe/Samara
67	America/St_Kitts	228	Asia/Kabul
68	America/St_Lucia	229	Indian/Kerguelen
69	America/Martinique	230	Asia/Tbilisi
70	America/Montserrat	231	Indian/Chagos
71	America/Puerto_Rico	232	Indian/Maldives
72	PRT	233	Asia/Dushanbe
73	America/Port_of_Spain	234	Asia/Ashkhabad
74	America/St_Vincent	235	Asia/Tashkent
75	America/Tortola	236	Asia/Karachi
76	America/St_Thomas	237	PLT
77	America/Caracas	238	Asia/Bishkek
78	Antarctica/Palmer	239	Asia/Aqtobe
79	Atlantic/Bermuda	240	Asia/Yekaterinburg
80	America/Cuiaba	241	Asia/Calcutta
81	America/Halifax	242	IST
82	Atlantic/Stanley	243	Asia/Katmandu
83	America/Thule	244	Antarctica/Mawson

Time zone setting	Time zone code	Time zone setting	Time zone code
84	America/Asuncion	245	Asia/Thimbu
85	America/Santiago	246	Asia/Colombo
86	America/St_Johns	247	Asia/Dacca
87	CNT	248	BST
88	America/Fortaleza	249	Asia/Almaty
89	America/Cayenne	250	Asia/Novosibirsk
90	America/Paramaribo	251	Indian/Cocos
91	America/Montevideo	252	Asia/Rangoon
92	America/Buenos_Aires	253	Indian/Christmas
93	AGT	254	Asia/Jakarta
94	America/Godthab	255	Asia/Phnom_Penh
95	America/Miquelon	256	Asia/Vientiane
96	America/Sao_Paulo	257	Asia/Saigon
97	BET	258	VST
98	America/Noronha	259	Asia/Bangkok
99	Atlantic/South_Georgia	260	Asia/Krasnoyarsk
100	Atlantic/Jan_Mayen	261	Antarctica/Casey
101	Atlantic/Cape_Verde	262	Australia/Perth
102	America/Scoresbysund	263	Asia/Brunei
103	Atlantic/Azores	264	Asia/Hong_Kong
104	Africa/Ouagadougou	265	Asia/Ujung_Pandang
105	Africa/Abidjan	266	Asia/Macao
106	Africa/Accra	267	Asia/Kuala_Lumpur
107	Africa/Banjul	268	Asia/Manila
108	Africa/Conakry	269	Asia/Singapore
109	Africa/Bissau	270	Asia/Taipei
110	Atlantic/Reykjavik	271	Asia/Shanghai
111	Africa/Monrovia	272	CTT
112	Africa/Casablanca	273	Asia/Ulan_Bator
113	Africa/Timbuktu	274	Asia/Irkutsk
114	Africa/Nouakchott	275	Asia/Jayapura
115	Atlantic/St_Helena	276	Asia/Pyongyang
116	Africa/Freetown	277	Asia/Seoul
117	Africa/Dakar	278	Pacific/Palau
118	Africa/Sao_Tome	279	Asia/Tokyo

Time zone setting	Time zone code	Time zone setting	Time zone code
119	Africa/Lome	280	JST
120	GMT	281	Asia/Yakutsk
121	UTC	282	Australia/Darwin
122	Atlantic/Faeroe	283	ACT
123	Atlantic/Canary	284	Australia/Adelaide
124	Europe/Dublin	285	Australia/Broken_Hill
125	Europe/Lisbon	286	Australia/Hobart
126	Europe/London	287	Antarctica/ DumontDUrville
127	Africa/Luanda	288	Pacific/Truk
128	Africa/Porto-Novo	289	Pacific/Guam
129	Africa/Bangui	290	Pacific/Saipan
130	Africa/Kinshasa	291	Pacific/Port_Moresby
131	Africa/Douala	292	Australia/Brisbane
132	Africa/Libreville	293	Asia/Vladivostok
133	Africa/Malabo	294	Australia/Sydney
134	Africa/Niamey	295	AET
135	Africa/Lagos	296	Australia/Lord_Howe
136	Africa/Ndjamena	297	Pacific/Ponape
137	Africa/Tunis	298	Pacific/Efate
138	Africa/Algiers	299	Pacific/Guadalcanal
139	Europe/Andorra	300	SST
140	Europe/Tirane	301	Pacific/Noumea
141	Europe/Vienna	302	Asia/Magadan
142	Europe/Brussels	303	Pacific/Norfolk
143	Europe/Zurich	304	Pacific/Kosrae
144	Europe/Prague	305	Pacific/Tarawa
145	Europe/Berlin	306	Pacific/Majuro
146	Europe/Copenhagen	307	Pacific/Nauru
147	Europe/Madrid	308	Pacific/Funafuti
148	Europe/Gibraltar	309	Pacific/Wake
149	Europe/Budapest	310	Pacific/Wallis
150	Europe/Rome	311	Pacific/Fiji
151	Europe/Vaduz	312	Antarctica/McMurdo
152	Europe/Luxembourg	313	Asia/Kamchatka

Time zone setting	Time zone code	Time zone setting	Time zone code
153	Africa/Tripoli	314	Pacific/Aucklandoo
154	Europe/Monaco	315	NST
155	Europe/Malta	316	Pacific/Chatham
156	Africa/Windhoek	317	Pacific/Enderbury
157	Europe/Amsterdam	318	Pacific/Tongatapu
158	Europe/Oslo	319	Asia/Anadyr
159	Europe/Warsaw	320	Pacific/Kiritimati
160	Europe/Stockholm		

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, you should ensure that the settings are made at the *local machine* level.

Services can continue to run under a special account when you log off a 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, click Read Only Mode on the Options menu in the registry editor.

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

```
SOFTWARE
   Sybase
      Adaptive Server IQ
```

Registry settings on installation

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

- Current version – In the Adaptive Server IQ registry, this entry holds the version number. For example:

  ```
  CurrentVersion:REG_SZ:12.7.0
  ```

- Description – In the Adaptive Server IQ registry, this entry holds the product name. For example:

  ```
  Description:REG_SZ:Adaptive Server IQ
  ```

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

  ```
  Location:REG_SZ:C:\Program Files\Sybase
  \ASIQ-12_7
  ```

- Install date – In the Adaptive Server IQ\12.7 registry, this entry holds the date the software was installed. For example:

  ```
  InstallDate:REG_SZ:10-20-2004
  ```

- Install type – In the Adaptive Server IQ\12.7 registry, this entry holds the type of installation. For example:

  ```
  InstallType:REG_SZ:Server
  ```

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

CHAPTER 2

Database Options

About this chapter

This chapter describes the database and DBISQL options you can set to customize and modify database behavior.

Contents

Introduction to database options

Database options control many aspects of database behavior. For example, you can use database options for the purposes such as the following:

- Compatibility – lets you control how much like Adaptive Server Enterprise your Sybase IQ database operates, and whether SQL that does not conform to SQL92 generates errors.
- Error handling – lets you control what happens when errors, such as dividing by zero or overflow errors, occur.
- Concurrency and transactions – lets you control the degree of concurrency and details of COMMIT behavior using options.

Setting options

You set options with the SET OPTION statement. It has the following general syntax:

SET [**EXISTING**] [**TEMPORARY**] **OPTION**
... [*userid*. | **PUBLIC**.]*option-name* = [*option-value*]

Specify a user ID or group name to set the option only for that user or group. Every user belongs to the PUBLIC group. If no user ID or group is specified, the option change is applied to the currently logged on user ID that issued the SET OPTION statement.

For example, the following statement applies an option change to the user DBA, if DBA is the user that issues it:

```
SET OPTION login_mode = mixed
```

The following statement applies a change to the PUBLIC user ID, a user group to which all users belong.

```
SET OPTION Public.login_mode = standard
```

Note For all database options that accept integer values, Sybase IQ truncates any decimal *option-value* setting to an integer value. For example, the value 3.8 is truncated to 3.

The maximum length of *option-value* when set to a string is 127 bytes.

Warning! Do not change option settings while fetching rows.

For more information, see the SET OPTION statement on page 647.

Finding option settings

You can obtain a list of option settings, or the values of individual options, in a variety of ways.

Getting a list of option values

- For the connected user, the sp_iqcheckoptions stored procedure displays a list of the current value and the default value of database options that have been changed from the default. sp_iqcheckoptions considers all Sybase IQ and ASA database options. Sybase IQ modifies some ASA 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.

  sp_iqcheckoptions also lists server start-up options that have been changed from the default values.

  When a DBA runs sp_iqcheckoptions, he or she 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 start-up options.

  The sp_iqcheckoptions stored procedure requires no parameters. In Interactive SQL, run the following command:

  ```
  sp_iqcheckoptions
  ```

  For more information, see “sp_iqcheckoptions procedure” on page 749.

  The system table DBA.SYSOPTIONDEFAULTS contains all of the names and default values of the Sybase IQ and ASA options. You can query this table to see all option default values.

- Current option settings for your connection are available as a subset of connection properties. You can list all connection properties using the sa_conn_properties system procedure.

  ```
  call sa_conn_properties
  ```

  To order this list, you can call sa_conn_properties_by_name.

For more information, see the section “sa_conn_properties_by_name system procedure” on page 857.

- In Interactive SQL, the SET statement with no arguments lists the current setting of options.

  ```
  SET
  ```

- In Sybase Central, right-click a database and select Options from the submenu.
- Use the following query on the SYSOPTIONS system view:

  ```
  SELECT *
  FROM SYSOPTIONS
  ```

  This shows all PUBLIC values, and those USER values that have been explicitly set.

Getting individual option values

You can obtain a single setting using the connection_property system function. For example, the following statement reports the value of the Ansinull option:

```
SELECT connection_property ('Ansinull')
```

Scope and duration of database options

You can set options at three levels of scope: public, user, and temporary.

Temporary options take precedence over user and public settings. User-level options take precedence over public settings. If you set a user-level option for the current user, the corresponding temporary option is set as well.

Some options, such as COMMIT behavior, are database-wide in scope. Setting these options requires DBA permissions. Other options, such as ISOLATION_LEVEL, can also be applied to only the current connection, and need no special permissions.

Changes to option settings take place at different times, depending on the option. Changing a global option such as RECOVERY_TIME takes place the next time the server is started. The following list contains some of the options that take effect after the server is restarted.

Database options that require restarting the server:

CACHE_PARTITIONS

CHECKPOINT_TIME

DISK_STRIPING

MAIN_CACHE_MEMORY_MB

Database options that require restarting the server:

MAIN_RESERVED_DBSPACE_MB

OS_FILE_CACHE_BUFFERING

OUT_OF_DISK_MESSAGE_REPEAT

OUT_OF_DISK_WAIT_TIME

PREFETCH_BUFFER_LIMIT

PREFETCH_BUFFER_PERCENT

RECOVERY_TIME

TEMP_CACHE_MEMORY_MB

TEMP_RESERVED_DBSPACE_MB

Options that affect only the current connection generally take place immediately. You can change option settings in the middle of a transaction, for example.

Warning! Changing options when a cursor is open can lead to unreliable results. For example, changing DATE_FORMAT might not change the format for the next row when a cursor is opened. Depending on the way the cursor is being retrieved, it might take several rows before the change works its way to the user.

Setting temporary options

Adding the TEMPORARY keyword to the SET OPTION statement changes the duration of the change. Ordinarily an option change is permanent: it will not change until it is explicitly changed using the SET OPTION statement.

When the SET TEMPORARY OPTION statement is executed, the new option value takes effect only for the current connection, and only for the duration of the connection.

When the SET TEMPORARY OPTION is used to set a PUBLIC option, the change is in place for as long as the database is running. When the database is shut down, Temporary options for the PUBLIC user ID revert back to their permanent value.

Setting an option for the PUBLIC user ID temporarily offers a security advantage. For example, when the LOGIN_MODE option is enabled the database relies on the login security of the system on which it is running. Enabling it temporarily means that a database relying on the security of a Windows domain will not be compromised if the database is shut down and copied to a local machine. In this case, the LOGIN_MODE option reverts to its permanent value, which could be Standard, a mode where integrated logins are not permitted.

Setting public options

Only users with DBA privileges have the authority to set an option for the PUBLIC user ID.

Changing the value of an option for the PUBLIC user ID sets the value of the option for all users who have not set their own value. An option value cannot be set for an individual user ID unless there is already a PUBLIC user ID setting for that option.

Deleting option settings

If *option-value* is omitted, the specified option setting is deleted from the database. If *option-value* was a personal option setting, the value reverts back to the PUBLIC setting. If a TEMPORARY option is deleted, the option setting reverts back to the permanent setting.

For example, the following statement resets the ANSINULL option to its default value:

```
SET OPTION ANSINULL =
```

If you incorrectly type the name of an option when you are setting the option, the incorrect name is saved in the SYSOPTION table. You can remove the incorrectly typed name from the SYSOPTION table by setting the option PUBLIC with an equality after the option name and no value:

```
SET OPTION PUBLIC.a_mistyped_name=;
```

For example, if you set an option and incorrectly type the name, you can verify that the option was saved by selecting from the SYSOPTIONS view:

```
SET OPTION PUBLIC.a_mistyped_name='ON';
SELECT * FROM SYSOPTIONS ORDER BY 2;
```

user_name	option	setting
PUBLIC	a_mistyped_name	ON
PUBLIC	Abort_On_Error_File	
PUBLIC	Abort_On_Error_Line	0
PUBLIC	Abort_On_Error_Number	0
...		

You can remove the incorrectly typed option by setting it to no value, then verify that the option is removed:

```
SET OPTION PUBLIC.a_mistyped_name=;
```

```
SELECT * FROM SYSOPTIONS ORDER BY 2;
```

user_name	option	setting
PUBLIC	Abort_On_Error_File	
PUBLIC	Abort_On_Error_Line	0
PUBLIC	Abort_On_Error_Number	0
...		

Option classification

Sybase IQ provides many options. It is convenient to divide them into a few general classes. The classes of options are:

- General database options
- Transact-SQL compatibility database options
- Interactive SQL (DBISQL) options

Note Each class of options is listed in a separate table in the following sections.

Initial option settings

Connections to Sybase IQ can be made through the TERANODE Design Suite (TDS) protocol—Open Client and jConnect™ for JDBC™ connections—or through the Sybase IQ protocol—ODBC, Embedded SQL.

If users have both TDS and the Sybase IQ-specific protocol, you can configure their initial settings using stored procedures. As it is shipped, Sybase IQ uses this method to set Open Client connections and jConnect connections to reflect default Adaptive Server Enterprise behavior.

The initial settings are controlled using the LOGIN_PROCEDURE option. This option names a stored procedure to run when users connect. The default setting is to use the sp_iq_process_login system stored procedure, which checks whether the user is permitted to log in, and then calls the sp_login_environment system procedure. You can change this behavior.

In its turn, sp_login_environment checks to see if the connection is being made over TDS. If it is, it calls the sp_tsql_environment procedure, which sets several options to new default values for the current connection.

For more information, including exceptions, see “LOGIN_PROCEDURE option” on page 106, or the sp_iq_process_login, sp_login_environment, and sp_tsql_environment system procedures in Chapter 10, “System Procedures.”

General database options

Table 2-1 lists database-specific options, their allowed values, and their default settings.

See the sections “Transact-SQL compatibility options” on page 35 and “DBISQL options” on page 37 for lists of the other classes of options.

Note There are additional internal options not listed in this table that Sybase Technical Support might ask you to use.

Table 2-1: General database options

OPTION	VALUES	DEFAULT
AGGREGATION_PREFERENCE	-3 to 3	0
APPEND_LOAD	ON, OFF	OFF
AUDITING	ON, OFF	OFF
BIT_VECTOR_PINNABLE_CACHE_PERCENT*	0 – 100	40
BLOCKING	OFF	OFF
BT_PREFETCH_MAX_MISS	0 – 1000	2
BT_PREFETCH_SIZE	0 – 100	10
CACHE_PARTITIONS	power of 2, 0 to 64	0
CHECKPOINT_TIME	number of minutes	60
CIS_ROWSET_SIZE	integer	50
CONVERSION_MODE	0, 1	0
CONVERT_HG_TO_1242	ON, OFF	OFF
CONVERT_VARCHAR_TO_1242	ON, OFF	OFF
COOPERATIVE_COMMIT_TIMEOUT	integer	250
COOPERATIVE_COMMITS	ON, OFF	ON
CURSOR_WINDOW_ROWS	20 – 100000	200
DATE_FIRST_DAY_OF_WEEK	0 – 6	0
DATE_FORMAT	string	'YYYY-MM-DD'
DATE_ORDER	'YMD', 'DMY', 'MDY'	'YMD'
DBCC_LOG_PROGRESS	ON, OFF	OFF

OPTION	VALUES	DEFAULT
DBCC_PINNABLE_CACHE_PERCENT	0 – 100	50
DDL_OPTIONS2	0 – 3	0
DEBUG_MESSAGES	ON, OFF	OFF
DEDICATED_TASK	ON, OFF	OFF
DEFAULT_HAVING_SELECTIVITY	0 – 100	0
DEFAULT_LIKE_MATCH_SELECTIVITY	0 – 100	15
DEFAULT_LIKE_RANGE_SELECTIVITY	0 – 100	15
DELAYED_COMMIT_TIMEOUT	integer	500
DELAYED_COMMITS	OFF	OFF
DISABLE_RI_CHECK	ON, OFF	OFF
DISK_STRIPING	ON, OFF	ON
DISK_STRIPING_PACKED	ON, OFF	ON
EARLY_PREDICATE_EXECUTION	ON, OFF	ON
ESCAPE_CHARACTER	ON, OFF	ON
ENABLED_ORDERED_PUSHDOWN_INSERTION	ON, OFF	ON
EXTENDED_JOIN_SYNTAX	ON, OFF	ON
FLATTEN_SUBQUERIES	ON, OFF	OFF
FORCE_DROP	ON, OFF	OFF
FORCE_NO_SCROLL_CURSORS	ON, OFF	OFF
FORCE_UPDATABLE_CURSORS	ON, OFF	OFF
FPL_EXPRESSION_MEMORY_KB	0 – 20000	1024
FP_PREDICATE_WORKUNIT_PAGES	integer	400
FP_PREFETCH_SIZE	0 – 100	10
GARRAY_FILL_FACTOR_PERCENT	0 – 1000	25
GARRAY_INSERT_PREFETCH_SIZE	0 – 100	3
GARRAY_RO_PREFETCH_SIZE	0 – 100	10
HASH_PINNABLE_CACHE_PERCENT*	0 – 100	20
HASH_THRASHING_PERCENT	0 – 100	10
HG_DELETE_METHOD	0 – 3	0
HG_SEARCH_RANGE	integer	10
IDENTITY_ENFORCE_UNIQUENESS	ON, OFF	OFF
IDENTITY_INSERT	string'	" (empty string)
INDEX_ADVISOR	ON, OFF	OFF
INDEX_PREFERENCE	-10 – 10	0
INFER_SUBQUERY_PREDICATES	ON, OFF	OFF
IN_SUBQUERY_PREFERENCE	-3 – 3	0
IQGOVERN_MAX_PRIORITY	1 – 3	2

OPTION	VALUES	DEFAULT
IQGOVERN_PRIORITY	1 – 3	2
IQGOVERN_PRIORITY_TIME	1 – 1,000,000 seconds	0 (disabled)
IQMSG_LENGTH_MB	0 – 2047	0
ISOLATION_LEVEL	0, 1, 2, 3	0
JAVA_HEAP_SIZE	integer	1000000
JAVA_NAMESPACE_SIZE	integer	4000000
JOIN_EXPANSION_FACTOR	0 – 100	30
JOIN_OPTIMIZATION	ON, OFF	ON
JOIN_PREFERENCE	-7 – 7	0
JOIN_SIMPLIFICATION_THRESHOLD	1 – 64	15
LARGE_DOUBLES_ACCUMULATOR	ON, OFF	OFF
LF_BITMAP_CACHE_KB	1 – 8	4
LOAD_MEMORY_MB	0 – 2000	0
LOCAL_KB_PER_STRIPE	integer > 0 in KB	1
LOAD_ZEROLENGTH_ASNULL	ON, OFF	OFF
LOCAL_RESERVED_DBSPACE_MB	integer > 0 in MB	200
LOG_CONNECT	ON, OFF	ON
LOG_CURSOR_OPERATIONS	ON, OFF	OFF
LOGIN_MODE	STANDARD, MIXED, INTEGRATED	STANDARD
LOGIN_PROCEDURE	string	sp_iq_process_login
MAIN_CACHE_MEMORY_MB	1 – 4194303	16
MAIN_KB_PER_STRIPE	integer > 0 in KB	1
MAIN_RESERVED_DBSPACE_MB	integer > 0 in MB	200
MAX_CARTESIAN_RESULT	integer	10000000
MAX_CLIENT_NUMERIC_PRECISION	0 – 126	0
MAX_CLIENT_NUMERIC_SCALE	0 – 126	0
MAX_CUBE_RESULT	0 – 250000000	10000000
MAX_CURSOR_COUNT	integer	50
MAX_HASH_ROWS	integer to 250000000	2500000
MAX_IQ_THREADS_PER_CONNECTION	3 – 1000	72
MAX_IQ_THREADS_PER_TEAM	1 – 1000	48
MAX_JOIN_ENUMERATION	1 – 64	15
MAX_QUERY_PARALLELISM	integer <= # CPUs	24
MAX_QUERY_TIME	$0 – 2^{32} - 1$	0 (disabled)
MAX_STATEMENT_COUNT	integer	100
MAX_WARNINGS	integer	$2^{64} - 1$

OPTION	VALUES	DEFAULT
MINIMIZE_STORAGE	ON, OFF	OFF
MIN_NLPDJ_FILTERED_PPM	1 – 1000000	2500
MIN_NLPDJ_TABLE_SIZE	1 – 4294967295	10000
MIN_PASSWORD_LENGTH	integer >= 0	0 characters
MIN_SMPDJ_OR_HPDJ_FILTERED_PPM	1 – 1000000	2500
MIN_SMPDJ_OR_HPDJ_FILTERED_SIZE	1 – 4294967295	25000
MIN_SMPDJ_OR_HPDJ_INDIRECT_SIZE	1 – 4294967295	500000
MIN_SMPDJ_OR_HPDJ_TABLE_SIZE	1 – 4294967295	100000
MONITOR_OUTPUT_DIRECTORY	string	*database directory*
MPX_GLOBAL_TABLE_PRIV	ON, OFF	OFF
MPX_LOCAL_SPEC_PRIV	0 to 63	0
NOEXEC	ON, OFF	OFF
NON_ANSI_NULL_VARCHAR	ON, OFF	OFF
NOTIFY_MODULUS	integer	100000
ODBC_DISTINGUISH_CHAR_AND_VARCHAR	ON, OFF	OFF
ON_CHARSET_CONVERSION_FAILURE	string	IGNORE
OS_FILE_CACHE_BUFFERING	ON, OFF	OFF
OUT_OF_DISK_MESSAGE_REPEAT	integer	120
OUT_OF_DISK_WAIT_TIME	integer	30
PARALLEL_GBH_ENABLED	ON, OFF	ON
PARALLEL_GBH_MIN_ROWS_PER_UNIT	0 – 4294967295	3000000
PARALLEL_GBH_UNITS	0 – 100	0
PRECISION	126	126
PREFETCH	ON, OFF	ON
PREFETCH_BUFFER_LIMIT	integer	0
PREFETCH_BUFFER_PERCENT	0 – 100	40
PREFETCH_GARRAY_PERCENT	0 – 100	60
PREFETCH_SORT_PERCENT	0 – 100	50
PRESERVE_SOURCE_FORMAT	ON, OFF	ON
QUERY_DETAIL	ON, OFF	OFF
QUERY_NAME	string	" (empty string)
QUERY_PLAN	ON, OFF	ON
QUERY_PLAN_AFTER_RUN	ON, OFF	OFF
QUERY_PLAN_AS_HTML	ON, OFF	OFF
QUERY_PLAN_AS_HTML_DIRECTORY	string	" (empty string)
QUERY_ROWS_RETURNED_LIMIT	integer	0
QUERY_TEMP_SPACE_LIMIT	integer	2000

OPTION	VALUES	DEFAULT
QUERY_TIMING	ON, OFF	OFF
RECOVERY_TIME	number of minutes	2
RETURN_DATE_TIME_AS_STRING	ON, OFF	OFF
ROW_COUNT	integer	0
SCALE	0 – 126	38
SIGNIFICANTDIGITSFORDOUBLEEQUALITY	0 – 15	0
SORT_PHASE1_HELPERS	integer	3
SORT_PINNABLE_CACHE_PERCENT*	0 – 100	20
SUBQUERY_PLACEMENT_PREFERENCE	-1 – 1	0
SUPPRESS_TDS_DEBUGGING	ON, OFF	OFF
SWEEPER_THREADS_PERCENT	1 to 40	10
TDS_EMPTY_STRING_IS_NULL	ON, OFF	OFF
TEMP_CACHE_MEMORY_MB	1 – 4194303	12
TEMP_DISK_PER_STRIPE	integer > 0 in KB	1
TEMP_EXTRACT_APPEND	ON, OFF	OFF
TEMP_EXTRACT_BINARY	ON, OFF	OFF
TEMP_EXTRACT_COLUMN_DELIMITER	string	','
TEMP_EXTRACT_DIRECTORY	string	'' (empty string)
TEMP_EXTRACT_NAME1 – TEMP_EXTRACT_NAME8	string	'' (empty string)
TEMP_EXTRACT_NULL_AS_EMPTY	ON, OFF	OFF
TEMP_EXTRACT_NULL_AS_ZERO	ON, OFF	OFF
TEMP_EXTRACT_QUOTE	string	'' (empty string)
TEMP_EXTRACT_QUOTES	ON, OFF	OFF
TEMP_EXTRACT_QUOTES_ALL	ON, OFF	OFF
TEMP_EXTRACT_ROW_DELIMITER	string	'' (empty string)
TEMP_EXTRACT_SIZE1 – TEMP_EXTRACT_SIZE8	AIX & HP-UX: 0 – 64GB Sun Solaris: & Linux 0 – 512GB Windows: 0 – 128GB	0
TEMP_EXTRACT_SWAP	ON, OFF	OFF
TEMP_KB_PER_STRIPE	integer > 0 in KB	1
TEMP_RESERVED_DBSPACE_MB	integer > 0 in MB	200
TEMP_SPACE_LIMIT_CHECK	ON, OFF	OFF
TIME_FORMAT	string	'HH:NN:ss.SSS'

OPTION	VALUES	DEFAULT
TIMESTAMP_FORMAT	string	'YYYY-MM-DD HH:NN:ss.SSS'
TRIM_PARTIAL_MBC	ON, OFF	OFF
TRUNCATE_WITH_AUTO_COMMIT	ON, OFF	ON
USER_RESOURCE_RESERVATION	integer	1
VERIFY_PASSWORD_FUNCTION	string	" (empty string)
WASH_AREA_BUFFERS_PERCENT	1 – 100	20
WAIT_FOR_COMMIT	ON, OFF	OFF

Data extraction options

The data extraction facility allows you to extract data from a database by redirecting the output of a SELECT statement from the standard interface to one or more disk files or named pipes. Several database options listed in Table 2-1 (TEMP_EXTRACT_...) are used to control this feature. For details on the use of these options, see Data extraction options in Chapter 7, “Moving Data In and Out of Databases” in the *Sybase IQ System Administration Guide*.

Transact-SQL compatibility options

The following options allow Sybase IQ behavior to be compatible with Adaptive Server Enterprise, or to both support old behavior and allow ISO SQL92 behavior.

For further compatibility with Adaptive Server Enterprise, you can set some of these options set for the duration of the current connection using the Transact-SQL SET statement instead of the Sybase IQ SET OPTION statement. For a listing of such options, see the SET statement on page 641.

Default settings

The default setting for some of these options differs from the Adaptive Server Enterprise default setting. To ensure compatible behavior, you should explicitly set the options.

When a connection is made using the Open Client or JDBC interfaces, some option settings are explicitly set for the current connection to be compatible with Adaptive Server Enterprise. These options are listed in Table 2-2.

For information on how the settings are made, see Chapter 10, “System Procedures.”

Table 2-2: Transact-SQL options set explicitly for ASE compatibility

Option	ASE-compatible setting
ALLOW_NULLS_BY_DEFAULT	OFF
ANSINULL	OFF
CHAINED	OFF
CONTINUE_AFTER_RAISERROR	ON
DATE_FORMAT	YYYY-MM-DD
DATE_ORDER	MDY
ESCAPE_CHARACTER	OFF
FLOAT_AS_DOUBLE	ON
ISOLATION_LEVEL	1
ON_TSQL_ERROR	CONDITIONAL
QUOTED_IDENTIFIER	OFF
TIME_FORMAT	HH:NN:SS.SSS
TIMESTAMP_FORMAT	YYYY-MM-DD HH:NN:SS.SSS
TSQL_HEX_CONSTANT	ON
TSQL_VARIABLES	OFF

List of options

Table 2-3 lists the compatibility options, their allowed values, and their default settings.

See “General database options” on page 30 and “DBISQL options” on page 37 for lists of the other classes of options.

Table 2-3: Transact-SQL compatibility options

Option	Values	Default
ALLOW_NULLS_BY_DEFAULT	ON, OFF	ON
ANSI_BLANKS*		
ANSI_CLOSE_CURSORS_ON_ROLLBACK	ON	ON
ANSI_INTEGER_OVERFLOW*		
ANSI_PERMISSIONS	ON, OFF	ON
ANSINULL	ON, OFF	ON
ANSI_UPDATE_CONSTRAINTS	OFF, CURSORS, STRICT	CURSORS
ASE_BINARY_DISPLAY	ON, OFF	ON
ASE_FUNCTION_BEHAVIOR	ON, OFF	OFF
AUTOMATIC_TIMESTAMP	OFF	OFF
CHAINED	ON, OFF	ON
CLOSE_ON_ENDTRANS	ON	ON
CONTINUE_AFTER_RAISEERROR	ON, OFF	ON
CONVERSION_ERROR	ON, OFF	ON

Option	Values	Default
DIVIDE_BY_ZERO_ERROR	ON, OFF	ON
ESCAPE_CHARACTER*	ON	ON
FIRE_TRIGGERS*		
FLOAT_AS_DOUBLE	ON, OFF	OFF
NEAREST_CENTURY	0 – 100	50
NON_KEYWORDS	Comma-separated keywords list	No keywords turned off
ON_TSQL_ERROR	STOP, CONTINUE, CONDITIONAL	CONDITIONAL
PERCENT_AS_COMMENT	ON, OFF	ON
QUERY_PLAN_ON_OPEN*		
QUOTED_IDENTIFIER	ON, OFF	ON
RI_TRIGGER_TIME*		
SQL_FLAGGER_ERROR_LEVEL	E, I, F, W	W
SQL_FLAGGER_WARNING_LEVEL	E, I, F, W	W
STRING_RTRUNCATION	ON, OFF	OFF
TEXTSIZE*		
TSQL_HEX_CONSTANT	ON, OFF	OFF
TSQL_VARIABLES	ON, OFF	OFF

Note An asterisk (*) next to the option name in Table 2-3 indicates an option currently not supported by Sybase IQ.

DBISQL options

These options change how DBISQL interacts with the database.

Syntax 1

SET OPTION
... [*userid*. | **PUBLIC**.]*option-name* = [*option-value*]

Syntax 2

SET PERMANENT

Syntax 3

SET

Parameters

userid:
identifier, *string* or *host-variable*

option-name:
identifier, *string* or *host-variable*

option-value:
host-variable (indicator allowed), *string*, *identifier*, or *number*

Description

SET PERMANENT (Syntax 2) stores all current DBISQL options in the SYSOPTIONS system table. These settings are automatically established every time DBISQL is started for the current user ID.

Syntax 3 is used to display all of the current option settings. If there are temporary options set for DBISQL or the database server, these are displayed; otherwise, the permanent option settings are displayed.

Table 2-4 lists the DBISQL options, their allowed values, and their default settings.

See “General database options” on page 30 and “Transact-SQL compatibility options” on page 35 for lists of the other classes of options.

Table 2-4: DBISQL options

Option	Values	Default
AUTO_COMMIT	ON, OFF	OFF
AUTO_REFETCH	ON, OFF	ON
BELL	ON, OFF	ON
COMMAND_DELIMITER	string	';'
COMMIT_ON_EXIT	ON, OFF	ON
DEFAULT_ISQL_ENCODING	identifier or string	empty string (use system code page)
ECHO	ON, OFF	ON
HEADINGS	ON, OFF	ON
INPUT_FORMAT*		
ISQL_COMMAND_TIMING	ON, OFF	ON
ISQL_ESCAPE_CHARACTER	single character	\ (backslash)
ISQL_FIELD_SEPARATOR	string	, (comma)
ISQL_LOG	file name	"
ISQL_QUOTE	string	' (single apostrophe)
NULLS	ON, OFF	NULL
ON_ERROR	STOP, CONTINUE, PROMPT, EXIT, NOTIFY_CONTINUE, NOTIFY_STOP, NOTIFY_EXIT	PROMPT
OUTPUT_FORMAT	ASCII, DBASEII, DBASEIII, EXCEL, FIXED, FOXPRO, HTML, LOTUS, SQL, XML,	ASCII
OUTPUT_LENGTH	Integer	0

Option	Values	Default
OUTPUT_NULLS	String	'NULL'
STATISTICS	0, 3, 4, 5, 6	3
TRUNCATION_LENGTH	integer	256

Note An asterisk (*) next to the option name in Table 2-4 indicates an option currently not supported by Sybase IQ.

Alphabetical list of options

This section lists options alphabetically.

Some option names are followed by an indicator in square brackets that indicates the class of the option. These indicators are as follows:

- [DBISQL] – The option changes how DBISQL interacts with the database.
- [TSQL] – The option allows Sybase IQ behavior to be made compatible with Adaptive Server Enterprise, or to both support old behavior and allow ISO SQL92 behavior.

AGGREGATION_PREFERENCE option

Function — Controls the choice of algorithms for processing an aggregate.

Allowed values — -3 – 3

Default — 0

Scope — DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description — For aggregation (GROUP BY, DISTINCT, SET functions) within a query, the Sybase IQ optimizer has a choice of several algorithms for processing the aggregate. This AGGREGATION_PREFERENCE option lets you override the optimizer's costing decision when choosing the algorithm. It does not override internal rules that determine whether an algorithm is legal within the query engine.

This option is normally used for internal testing and for manually tuning queries that the optimizer does not handle well. Only experienced DBAs should use it. Inform Sybase Technical Support if you need to set AGGREGATION_PREFERENCE, as setting this option might mean that a change to the optimizer is appropriate.

Table 2-5 describes the valid values and their actions for the AGGREGATION_PREFERENCE option.

Table 2-5: AGGREGATION_PREFERENCE values

Value	Action
0	Let the optimizer choose
1	Prefer aggregation with a sort
2	Prefer aggregation using IQ indexes
3	Prefer aggregation with a hash
-1	Avoid aggregation with a sort
-2	Avoid aggregation using IQ indexes
-3	Avoid aggregation with a hash

ALLOW_NULLS_BY_DEFAULT option [TSQL]

Function — Controls whether new columns created without specifying either NULL or NOT NULL are allowed to contain NULL values.

Allowed values — ON, OFF

Default — ON

OFF for Open Client and JDBC connections

Description — The ALLOW_NULLS_BY_DEFAULT option is included for Transact-SQL compatibility.

See also — Appendix A, “Compatibility with Other Sybase Databases.’

ANSI_CLOSE_CURSORS_ON_ROLLBACK option [TSQL]

Function — Controls whether cursors that were opened WITH HOLD are closed when a ROLLBACK is performed.

Allowed values — ON

Default — ON

Description

The ANSI SQL/3 standard requires all cursors be closed when a transaction is rolled back. This option forces that behavior and cannot be changed. The CLOSE_ON_ENDTRANS option overrides this option.

ANSI_PERMISSIONS option [TSQL]

Function

Controls permissions checking for DELETE and UPDATE statements.

Allowed values

ON, OFF

Default

ON

Description

With ANSI_PERMISSIONS ON, SQL92 permissions requirements for DELETE and UPDATE statements are checked. The default value is OFF in Adaptive Server Enterprise. Table 2-6 outlines the differences.

Table 2-6: Effect of ANSI_PERMISSIONS option

SQL statement	Permissions required with ANSI_PERMISSIONS OFF	Permissions required with ANSI_PERMISSIONS ON
UPDATE	UPDATE permission on the columns where values are being set	UPDATE permission on the columns where values are being set SELECT permission on all columns appearing in the WHERE clause. SELECT permission on all columns on the right side of the set clause.
DELETE	DELETE permission on table	DELETE permission on table. SELECT permission on all columns appearing in the WHERE clause.

The ANSI_PERMISSIONS option can be set only for the PUBLIC group. No private settings are allowed.

ANSINULL option [TSQL]

Function

Controls the interpretation of using = and != with NULL.

Allowed values

ON, OFF

Default: ON

Description: With ANSINULL ON, results of comparisons with NULL using '=' or '!=' are unknown. This includes results of comparisons implied by other operations such as CASE.

Setting ANSINULL to OFF allows comparisons with NULL to yield results that are not unknown, for compatibility with Adaptive Server Enterprise.

Note Unlike Adaptive Server Anywhere, Sybase IQ does *not* generate the warning “null value eliminated in aggregate function” (SQLSTATE=01003) for aggregate functions on columns containing NULL values.

ANSI_UPDATE_CONSTRAINTS option

Function: Controls the range of updates that are permitted.

Allowed values: OFF, CURSORS, STRICT

Default: CURSORS in new databases.

OFF in databases created before version 12.4.3.

Description: Sybase IQ provides several extensions that allow updates that are not permitted by the ANSI SQL standard. These extensions provide powerful, efficient mechanisms for performing updates. However, in some cases, they cause behavior that is not intuitive. This behavior might produce anomalies such as lost updates if the user application is not designed to expect the behavior of these extensions.

The ANSI_UPDATE_CONSTRAINTS option controls whether updates are restricted to those permitted by the SQL92 standard.

If the option is set to STRICT, the following updates are prevented:

- Updates of cursors containing JOINS
- Updates of columns that appear in an ORDER BY clause
- The FROM clause is not allowed in UPDATE statements.

If the option is set to CURSORS, these same restrictions are in place, but only for cursors. If a cursor is not opened with FOR UPDATE or FOR READ ONLY, the database server determines whether updates are permitted based on the SQL92 standard.

If the ANSI_UPDATE_CONSTRAINTS option is set to CURSORS or STRICT, cursors containing an ORDER BY clause default to FOR READ ONLY; otherwise, they continue to default to FOR UPDATE.

Example

The following code has a different effect, depending on the setting of ANSI_UPDATE_CONSTRAINTS.

```
create table mmg (a char(3));
create table mmg1 (b char(3));

insert into mmg values ('001');
insert into mmg values ('002');
insert into mmg values ('003')
insert into mmg1 values ('003');
select * from mmg;
select * from mmg1;
```

Option 1: Set ANSI_UPDATE_CONSTRAINTS to STRICT:

```
set option public.Ansi_update_constraints = 'strict';
DELETE MMG FROM MMG1 WHERE A=B;
```

This results in an error indicating that the attempted update operation is not allowed.

Option 2: Set ANSI_UPDATE_CONSTRAINTS to CURSORS or OFF:

```
set option public.Ansi_update_constraints = 'CURSORS';
// or 'OFF'
DELETE MMG FROM MMG1 WHERE A=B;
```

In this case, the deletion should complete without the error.

See also

UPDATE statement on page 661.

APPEND_LOAD option

Function

Helps reduce space usage from versioned pages.

Allowed values

ON, OFF

Default

OFF

Scope

DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

The APPEND_LOAD option applies to LOAD, INSERT...SELECT, and INSERT...VALUES statements. It takes effect on the next LOAD, INSERT...SELECT, or INSERT...VALUES statement.

When the APPEND_LOAD option is OFF, Sybase IQ reuses row IDs from deleted rows. Setting this option ON appends new data to the end of the table.

ASE_BINARY_DISPLAY option

Function	Specifies that the display of Sybase IQ binary columns is consistent with the display of Adaptive Server Enterprise binary columns.
Allowed values	ON, OFF
Default	ON
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	The ASE_BINARY_DISPLAY option affects the output of the SELECT statement. This option affects only columns in the IQ Store. It does not affect variables, Catalog Store columns or Adaptive Server Anywhere columns. When this option is ON, Sybase IQ displays the column in readable ASCII format; for example, 0x1234567890abcdef). When this option is OFF, Sybase IQ displays the column as binary output (not ASCII). Set ASE_BINARY_DISPLAY OFF to support bulk copy operations (using iq_bcp) on binary data types.
See also	"Bulk Copy utility (iq_bcp)," Chapter 3, "Database Administration Utilities" in the *Sybase IQ Utility Guide*.

ASE_FUNCTION_BEHAVIOR option

Function	Specifies that output of Sybase IQ functions, including INTTOHEX and HEXTOINT, is consistent with the output of Adaptive Server Enterprise functions.
Allowed values	ON, OFF
Default	OFF
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

When the ASE_BEHAVIOR_FUNCTION option is ON, some of the Sybase IQ data type conversion functions, including HEXTOINT and INTTOHEX, return output that is consistent with the output of Adaptive Server Enterprise functions. The differences in the ASE and Sybase IQ output, with respect to formatting and length, exist because ASE primarily uses signed 32-bit as the default and Sybase IQ primarily uses unsigned 64-bit as the default.

Sybase IQ does not provide support for 64-bit integer, as ASE does not have a 64-bit integer data type.

For details on the behavior of the INTTOHEX and HEXTOINT functions when the ASE_FUNCTION_BEHAVIOR option is enabled, see "INTTOHEX function [Data type conversion]" on page 314 and "HEXTOINT function [Data type conversion]" on page 306 in Chapter 5, "SQL Functions."

Example

In this example, the HEXTOINT function returns a different value based on whether the ASE_FUNCTION_BEHAVIOR option is ON or OFF.

The HEXTOINT function returns `4294967287` with ASE_FUNCTION_BEHAVIOR OFF:

```
select hextoint('fffffff7') from iq_dummy
```

The HEXTOINT function returns `-9` with ASE_FUNCTION_BEHAVIOR ON:

```
select hextoint('fffffff7') from iq_dummy
```

See also

"HEXTOINT function [Data type conversion]" on page 306.

"INTTOHEX function [Data type conversion]" on page 314.

"CONVERSION_ERROR option [TSQL]" on page 53.

AUDITING option [database]

Function

Enables and disables auditing in the database.

Allowed values

ON, OFF

Default

OFF

Description

This option turns auditing on and off.

Auditing is the recording of detailed information about many events in the database in the transaction log. Auditing provides some security features, at the cost of some performance.

For the AUDITING option to work, you must set the AUDITING option to ON, and also specify which types of information you want to audit using the sa_enable_auditing_type system procedure. Auditing will not occur if either of the following are true:

- AUDITING is set to OFF
- Auditing options have been disabled

If you set AUDITING ON, and do not specify auditing options, all types of auditing information are recorded. Alternatively, you can choose to record any combination of the following: audit permission checks, connection attempts, DDL statements, public options, and triggers.

Can be set for the PUBLIC group only. Takes effect immediately. Requires DBA permissions to set this option.

See also

"sa_enable_auditing_type system procedure" on page 859.

AUTO_COMMIT option [DBISQL]

Function

Controls whether a COMMIT is performed after each statement.

Allowed values

ON, OFF

Default

OFF

Description

If AUTO_COMMIT is ON, a database COMMIT is performed after each successful statement. If the COMMIT fails, you have the option to execute additional SQL statements and perform the COMMIT again, or execute a ROLLBACK statement.

By default, a COMMIT or ROLLBACK is performed only when the user issues a COMMIT or ROLLBACK statement or a SQL statement that causes an automatic commit (such as the CREATE TABLE statement).

AUTO_COMMIT basically performs the same function as CHAINED, except that AUTO_COMMIT takes effect only if you are running DBISQL.

AUTO_REFETCH option [DBISQL]

Function

Controls whether query results are fetched again after deletes, updates, and inserts.

Allowed values

ON, OFF

Default	ON
Description	If AUTO_REFETCH is ON, then the current query results are refetched from the database after *any* INSERT, UPDATE or DELETE statement. Depending on how complicated the query is, this might take some time. For this reason, it can be turned OFF.

AUTOMATIC_TIMESTAMP option [TSQL]

Function	Controls interpretation of new columns with TIMESTAMP data type.
Allowed values	OFF
Default	OFF
Description	Controls whether any new columns with the TIMESTAMP data type that do not have an explicit default value defined are given a default value of the Transact-SQL timestamp value. Currently, Sybase IQ does not support this feature, so the default and only value allowed is OFF.

BELL option [DBISQL]

Function	Controls whether the bell sounds when an error occurs.
Allowed values	ON, OFF
Default	ON
Description	Set this option according to your preference.

BIT_VECTOR_PINNABLE_CACHE_PERCENT option

Function	Maximum percentage of a user's temp memory that a persistent bit-vector object can pin.
Allowed values	0 – 100
Default	40
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description — BIT_VECTOR_PINNABLE_CACHE_PERCENT controls the percentage of a user's temp memory allocation that any one persistent bit-vector object can pin in memory. It defaults to 40%, and should not generally be changed by users.

This option is primarily for use by Sybase Technical Support. If you change the value of BIT_VECTOR_PINNABLE_CACHE_PERCENT, do so with extreme caution; first analyze the effect on a wide variety of queries.

See also — "HASH_PINNABLE_CACHE_PERCENT option" on page 82.

"SORT_PINNABLE_CACHE_PERCENT option" on page 146.

BLOCKING option

Function — Controls the behavior in response to locking conflicts.

Allowed values — OFF

Default — OFF

Scope — Can be set for an individual connection or the PUBLIC group. Takes effect immediately.

Description — When BLOCKING is OFF, a transaction receives an error when it attempts a write operation and it is blocked by another transaction's read lock.

BT_PREFETCH_MAX_MISS option

Function — Controls the way Sybase IQ determines whether to continue prefetching B-tree pages for a given query.

Allowed values — 0 – 1000

Default — 2

Scope — Can be set for an individual connection or for the PUBLIC group. Takes effect immediately.

Description — Use only if instructed to do so by Sybase Technical Support. For queries that use HG (High_Group) indexes, Sybase IQ prefetches B-tree pages sequentially until it determines that prefetching is no longer useful. For some queries, it might turn off prefetching prematurely. Increasing the value of BT_PREFETCH_MAX_MISS makes it more likely that Sybase IQ continues prefetching, but also might increase I/O unnecessarily.

If queries using HG indexes run more slowly than expected, try gradually increasing the value of this option.

Experiment with different settings to find the one that gives the best performance. For most queries, useful settings are in the range of 1 to 10.

See also

"BT_PREFETCH_SIZE option" on page 49.

"PREFETCH_BUFFER_LIMIT option" on page 136.

BT_PREFETCH_SIZE option

Function

Restricts the size of the read-ahead buffer for the High_Group B-tree.

Allowed values

0 – 100. Setting to 0 disables B-tree prefetch.

Default

10

Scope

Can be set only for an individual user. Takes effect immediately.

Description

B-tree prefetch is activated by default for any sequential access to the High_Group index such as INSERT, large DELETE, range predicates, and DBCC (Database Consistency Checker commands).

This option limits the size of the read-ahead buffer for B-tree pages. Reducing prefetch size frees buffers, but also degrades performance at some point. Increasing prefetch size might have marginal returns. This option should be used in conjunction with the options PREFETCH_GARRAY_PERCENT, GARRAY_INSERT_PREFETCH_SIZE, and GARRAY_RO_PREFETCH_SIZE for non-unique High_Group indexes.

CACHE_PARTITIONS option

Function

Sets the number of partitions to be used for the main and temporary buffer caches.

Allowed values

0, 1, 2, 4, 8, 16, 32, 64:

Table 2-7: CACHE_PARTITIONS values

Value	Description
0	Sybase IQ computes the number of partitions automatically as *number_of_cpus/8*, rounded to the nearest power of 2, up to a maximum of 64.
1	1 partition only; this value disables partitioning.
2 – 64	Number of partitions; must be a power of 2.

Default

0 (Sybase IQ computes the number of partitions automatically).

Scope

Can be set for the PUBLIC group only. Takes effect for the current database the next time you start the database server.

Description

Partitioning the buffer cache can sometimes improve performance on systems with multiple CPUs by reducing lock contention. Normally you should rely on the value that Sybase IQ calculates automatically, which is based on the number of CPUs on your system. However, if you find that load or query performance in a multi-CPU configuration is slower than expected, you might be able to improve it by setting a different value for CACHE_PARTITIONS.

Both the number of CPUs and the platform can influence the ideal number of partitions. Experiment with different values to determine the best setting for your configuration.

The value you set for CACHE_PARTITIONS applies to both the main and temp buffer caches. The absolute maximum number of partitions is 64, for each buffer cache.

The -iqpartition server option sets the partition limit at the server level. If -iqpartition is specified at server start-up, it always overrides the CACHE_PARTITIONS setting.

The number of partitions does not affect other buffer cache settings. It also does not affect statistics collected by the IQ monitor; statistics for all partitions are rolled up and reported as a single value.

Example

In a system with 100 CPUs, if you do not set CACHE_PARTITIONS, Sybase IQ automatically sets the number of partitions to 16 as follows:

100 cpus/8 = 12, rounded to 16.

With this setting, there are 16 partitions for the main cache and 16 partitions for the temp cache.

In the same system with 100 CPUs, to explicitly set the number of partitions to 8, specify:

```
SET OPTION "PUBLIC".CACHE_PARTITIONS=8
```

See also

-iqpartition in Server command-line switches in the *Sybase IQ Utility Guide*.

"Managing lock contention" on page 495 in Chapter 10, "Transactions and Versioning" in the *Sybase IQ System Administration Guide*.

CHAINED option [TSQL]

Function

Controls transaction mode in the absence of a BEGIN TRANSACTION statement.

Allowed values

ON, OFF

OFF for Open Client and JDBC connections

Default

ON

Description

Controls the Transact-SQL transaction mode. In unchained mode (CHAINED = OFF) each statement is committed individually unless an explicit BEGIN TRANSACTION statement is executed to start a transaction. In chained mode (CHAINED = ON) a transaction is implicitly started before any data retrieval or modification statement. For Adaptive Server Enterprise, the default setting is OFF.

CHECKPOINT_TIME option

Function

Set the maximum length of time, in minutes, that the database server runs without doing a checkpoint.

Allowed values

Integer

Default

60

Scope

Can be set only for the PUBLIC group. Requires DBA permissions to set the option. You must shut down and restart the database server for the change to take effect.

Description

This option is used with the "RECOVERY_TIME option" on page 143 to decide when checkpoints should be done.

CIS_ROWSET_SIZE option

Function

Set the number of rows that are returned from remote servers for each fetch.

Allowed values	Integer
Default	50
Scope	Can be set for an individual connection or the PUBLIC group. Takes effect when a new connection is made to a remote server.
Description	This option sets the ODBC FetchArraySize value when you are using ODBC to connect to a remote database server.
See also	For information on remote data access, see Chapter 16, "Accessing Remote Data" in the *Sybase IQ System Administration Guide*.

CLOSE_ON_ENDTRANS option [TSQL]

Function	Controls closing of cursors at the end of a transaction.
Allowed values	ON
Default	ON
Description	When CLOSE_ON_ENDTRANS is set to ON (the default and only value allowed), cursors are closed at the end of a transaction. With this option set ON, it provides Transact-SQL compatible behavior.

COMMAND_DELIMITER option [DBISQL]

Function	Sets the string indicating the termination of a statement in DBISQL.
Allowed values	String
Default	Semicolon (;)
Description	If the command delimiter is set to a string beginning with a character that is valid in identifiers, the command delimiter must be preceded by a space.

COMMIT_ON_EXIT option [DBISQL]

Function	Controls behavior when DBISQL disconnects or terminates.
Allowed values	ON, OFF
Default	ON

Description — Controls whether a COMMIT or ROLLBACK is done when you leave DBISQL. When COMMIT_ON_EXIT is set to ON, a COMMIT is performed; otherwise a ROLLBACK is performed.

CONTINUE_AFTER_RAISERROR option [TSQL]

Function — Controls behavior following a RAISERROR statement.

Allowed values — ON, OFF

Default — ON

Description — The RAISERROR statement is used within procedures to generate an error. When the option is set to OFF, the execution of the procedure is stopped when the RAISERROR statement is encountered.

When the CONTINUE_AFTER_RAISERROR switch is ON, the RAISERROR statement no longer signals an execution-ending error. Instead, the RAISERROR status code and message are stored and the most recent RAISERROR is returned when the procedure completes. If the procedure that caused the RAISERROR was called from another procedure, the RAISERROR is not returned until the outermost calling procedure terminates.

Intermediate RAISERROR statuses and codes are lost after the procedure terminates. If, at return time, an error occurs along with the RAISERROR, then the error information is returned and the RAISERROR information is lost. The application can query intermediate RAISERROR statuses by examining @@error global variable at different execution points.

The setting of the CONTINUE_AFTER_RAISERROR option is used to control behavior following a RAISERROR statement *only* if the ON_TSQL_ERROR option is set to CONDITIONAL (the default). If you set the ON_TSQL_ERROR option to STOP or CONTINUE, the ON_TSQL_ERROR setting takes precedence over the CONTINUE_AFTER_RAISERROR setting.

See also — "ON_TSQL_ERROR option [TSQL]" on page 128.

CONVERSION_ERROR option [TSQL]

Function — Controls reporting of data type conversion failures on fetching information from the database.

Allowed values — ON, OFF

Default: ON

Description: This option controls whether data type conversion failures, when data is fetched from the database or inserted into the database, are reported by the database as errors (CONVERSION_ERROR set to ON), or as warnings (CONVERSION_ERROR set to OFF).

When CONVERSION_ERROR is set to ON, the SQLE_CONVERSION_ERROR error is generated.

If the option is set to OFF, the warning SQLE_CANNOT_CONVERT is produced. Each thread doing data conversion for a LOAD statement writes at most one warning message to the *.iqmsg* file. In order to write all data conversion warning messages to the*.iqmsg* file, you must set the DDL_OPTIONS2 option to 8 after setting CONVERSION_ERROR OFF. For example:

```
SET TEMPORARY OPTION CONVERSION_ERROR='OFF'
SET TEMPORARY OPTION DDL_OPTIONS2=8
```

If conversion errors are reported as warnings only, the NULL value is used in place of the value that could not be converted. In Embedded SQL, an indicator variable is set to -2 for the column or columns that cause the error.

CONVERSION_MODE option

Function: Restricts implicit conversion between binary data types (BINARY, VARBINARY, and LONG BINARY) and other non-binary data types (BIT, TINYINT, SMALLINT, INT, UNSIGNED INT, BIGINT, UNSIGNED BIGINT, CHAR, VARCHAR, and LONG VARCHAR) on various operations.

Allowed values: 0, 1

Default: 0

Scope: Can be set either publicly or temporarily. DBA permissions are not required to set this option.

Description: The default value of 0 maintains implicit conversion behavior prior to version 12.7. Setting CONVERSION_MODE to 1 restricts implicit conversion of binary data types to any other non-binary data type on INSERT, UPDATE, and in queries. The restrict binary conversion mode also applies to LOAD TABLE default values and CHECK constraint. The use of this option prevents implicit data type conversions of encrypted data that would result in semantically meaningless operations.

Implicit conversion restrictions

The CONVERSION_MODE option restrict binary mode value of 1 restricts implicit conversion for the following operations.

LOAD TABLE The restrict implicit binary conversion mode applies to LOAD TABLE with CHECK constraint or default value.

For example:

```
CREATE TABLE t3 (c1 INT,
          csi SMALLINT,
          cvb VARBINARY(2),
          CHECK (csi<cvb));
SET TEMPORARY OPTION CONVERSION_MODE = 1;
```

The following request:

```
LOAD TABLE t3(c1 ',', csi ',', cvb ',')
         FROM 't3.inp'
         QUOTES OFF ESCAPES OFF
         ROW DELIMITED BY '\n'
```

fails with the message:

```
"Invalid data type comparison in predicate
(t3.csi < t3.cvb), [-1001013] ['QFA13']"
```

INSERT The restrict implicit binary conversion mode applies to INSERT...SELECT, INSERT...VALUE, and INSERT...LOCATION.

For example:

```
CREATE TABLE t1 (c1 INT PRIMARY KEY,
          cbt  BIT NULL
          cti  TINYINT,
          csi  SMALLINT,
          cin  INTEGER,
          cui  UNSIGNED INTEGER,
          cbi  BIGINT,
          cub  UNSIGNED BIGINT,
          cch  CHAR(10),
          cvc  VARCHAR(10),
          cbn  BINARY(8),
          cvb  VARBINARY(8),
          clb  LONG BINARY,
          clc  LONG VARCHAR));

CREATE TABLE t2 (c1 INT PRIMARY KEY,
          cbt  BIT NULL,
          cti  TINYINT,
          csi  SMALLINT,
```

```
            cin  INTEGER,
            cui  UNSIGNED INTEGER,
            cbi  BIGINT,
            cub  UNSIGNED BIGINT,
            cch  CHAR(10),
            cvc  VARCHAR(10),
            cbn  BINARY(8),
            cvb  VARBINARY(8),
            clb  LONG BINARY,
            clc  LONG VARCHAR));

CREATE TABLE t4 (c1 INT, cin INT DEFAULT 0x31);

SET TEMPORARY OPTION CONVERSION_MODE = 1;
```

The following request:

```
INSERT INTO t1(c1, cvb) SELECT 99, cin FROM T2
WHERE c1=1
```

fails with the message:

```
"Unable to convert column 'cvb' to the requested
datatype (varbinary) from datatype (integer).
[-1013043] ['QCA43']"
```

The following request:

```
INSERT INTO t4 VALUES (1, DEFAULT)
```

fails with the message:

```
"Unable to convert column 'cin' to the requested
datatype (integer) from datatype (varbinary).
[-1013043] ['QCA43']"
```

UPDATE The restrict implicit binary conversion mode applies to the following types of UPDATE:

UPDATE SET VALUE FROM *expression* (including constant)
UPDATE SET VALUE FROM *other column*
UPDATE SET VALUE FROM *host variable*
JOIN UPDATE SET VALUE FROM *column of other table*

For example, the following request:

```
UPDATE t1 SET cbi=cbn WHERE c1=1
```

fails with the message:

```
"Unable to implicitly convert column 'cbi' to datatype
(bigint) from datatype (binary). [-1000187] ['QCB87']"
```

Positioned INSERT and positioned UPDATE via updatable cursor The restrict implicit binary conversion mode applies to the following types of INSERT and UPDATE via updatable cursor:

PUT *cursor-name* USING ... *host-variable*
Positioned UPDATE from another column
Positioned UPDATE from a constant
Positioned UPDATE from a host variable

For example, the following request:

```
BEGIN
    DECLARE curs SCROLL CURSOR FOR SELECT * FROM t1
    FOR UPDATE;
    OPEN curs WITH HOLD;
    FETCH curs;
    UPDATE t1 SET cbi=cbn WHERE CURRENT OF curs;
END
```

fails with the message:

```
"Unable to implicitly convert column 'cbn' to datatype
(bigint) from datatype (binary). [-1000187] ['QCB87']"
```

Queries The restrict implicit binary conversion mode applies to all aspects of queries in general.

1 **Comparison Operators**
When CONVERSION_MODE = 1, the restriction applies to the following operators:

=, !=, <, <=, >=, <>, !>, !<
BETWEEN .. AND
IN

used in a search condition for the following clauses:

WHERE clause
HAVING clause
CHECK clause
ON phrase in a join
IF/CASE expression

For example, the following query:

```
SELECT COUNT(*) FROM T1
WHERE cvb IN (SELECT csi FROM T2)
```

fails with the message:

```
"Invalid data type comparison in predicate (t1.cvb
IN (SELECT t1.csi ...)), [-1001013] ['QFA13']"
```

2 **String Functions**
When CONVERSION_MODE = 1, the restriction applies to the following string functions:

CHAR
CHAR_LENGTH
DIFFERENCE
LCASE
LEFT
LOWER
LTRIM
PATINDEX
RIGHT
RTRIM
SIMILAR
SORTKEY
SOUNDEX
SPACE
STR
TRIM
UCASE
UPPER

For example, the following query:

```
SELECT ASCII(cvb) FROM t1 WHERE c1=1
```

fails with the message:

```
"Data exception - data type conversion is not
possible. Argument to ASCII must be string,
[-1009145] ['QFA2E']"
```

The following functions allow either a string argument or a binary argument. When CONVERSION_MODE = 1, the restriction applies to mixed type arguments, that is, one argument is string and the other argument is binary.

INSERTSTR
LOCATE
REPLACE
STRING
STUFF

For example, the following query:

```
SELECT STRING(cvb, cvc) FROM t1 WHERE c1=1
```

where the column cvb is defined as VARBINARY and the column cvc is defined as VARCHAR, fails with the message:

```
"Data exception - data type conversion is not
possible. Arguments to STRING must be all binary or
all string, [-1009145] ['QFA2E']"
```

The restriction does *not* apply to the following string functions:

BIT_LENGTH
BYTE_LENGTH
CHARINDEX
LENGTH
OCTET_LENGTH
REPEAT
REPLICATE
SUBSTRING

3 **Arithmetic Operations and Functions**
When CONVERSION_MODE = 1, the restriction applies to the following operators used in arithmetic operations:

+, -, *, /

The restriction applies to the following bitwise operators used in bitwise expressions:

& (AND), | (OR), ^ (XOR)

The restriction also applies to integer arguments of the following functions:

ROUND
"TRUNCATE"
TRUNCNUM

For example, the following query:

```
SELECT ROUND(4.4, cvb) FROM t1 WHERE C1=1
```

fails with the message:

```
"Data exception - data type conversion is not
possible. Second Argument to ROUND cannot be
converted into an integer, [-1009145] ['QFA2E']"
```

4 **Integer Argument to Various Functions**
When CONVERSION_MODE = 1, the restriction applies to integer argument of the following functions:

ARGN
SUBSTRING
DATEADD
YMD

For example, the following query:

```
SELECT ARGN(cvb, csi, cti) FROM t1 WHERE c1=1
```

fails with the message:

```
"Data exception - data type conversion is not
possible. First Argument to ARGN cannot be converted
to an integer, [-1009145] ['QFA2E']"
```

5 **Analytical Functions, Aggregate Functions, and Numeric Functions**
When CONVERSION_MODE = 1, no further restriction applies to analytical functions, aggregate functions, and numeric functions that require numeric expressions as arguments.

See also

For more information on data type conversion, see Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

For more information on column encryption, see *Encrypted Columns in Sybase IQ*. *Users must be specifically licensed to use the encrypted column functionality of the* Sybase IQ *Encrypted Column Option.*

CONVERT_HG_TO_1242 option

Function — Converts pre-version 12.4.2 HG (High_Group) indexes to an improved format.

Allowed values — ON, OFF

Default — OFF

Scope — Can be set only for the PUBLIC group. Takes effect when you run sp_iqcheckdb in any mode.

Description — Improves read performance of queries against HG indexes.

Set this option and then run sp_iqcheckdb only once, and only for columns with HG indexes that were created before version 12.4.2.

CONVERT_VARCHAR_TO_1242 option

Function	Converts pre-version 12.4.2 VARCHAR data to compressed format.
Allowed values	ON, OFF
Default	OFF
Scope	Can be set only for the PUBLIC group. Takes effect when you run sp_iqcheckdb in any mode.
Description	Helps further compress data and improve performance, especially for databases with many variable character strings. Set this option and then run sp_iqcheckdb only once, and only for VARCHAR columns that were created before version 12.4.2.

COOPERATIVE_COMMIT_TIMEOUT option

Function	Governs when a COMMIT entry in the transaction log is written to disk.
Allowed values	Integer, in milliseconds
Default	250
Scope	Can be set for an individual connection or the PUBLIC group. Takes effect immediately.
Description	This option only has meaning when COOPERATIVE_COMMITS is set to ON. The database server waits for the specified number of milliseconds for other connections to fill a page of the log before writing to disk. The default setting is 250 milliseconds.

COOPERATIVE_COMMITS option

Function	Controls when commits are written to disk.
Allowed values	ON, OFF
Default	ON
Scope	Can be set for an individual connection or the PUBLIC group. Takes effect immediately.
Description	If COOPERATIVE_COMMITS is set to OFF, a COMMIT is written to disk as soon as the database server receives it, and the application is then allowed to continue.

If COOPERATIVE_COMMITS is set to ON, the default, the database server does not immediately write the COMMIT to the disk. Instead, it requires the application to wait for a maximum length set by the COOPERATIVE_COMMIT_TIMEOUT option for something else to put on the pages before the commit is written to disk.

Setting COOPERATIVE_COMMITS to ON, and increasing the COOPERATIVE_COMMIT_TIMEOUT setting increases overall database server throughput by cutting down the number of disk I/Os, but at the expense of a longer turnaround time for each individual connection.

CURSOR_WINDOW_ROWS option

Function	Defines the number of cursor rows to buffer.
Allowed values	20 – 100000
Default	200
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	When an application opens a cursor, Sybase IQ creates a FIFO (first-in, first-out) buffer to hold the data rows generated by the query. CURSOR_WINDOW_ROWS defines how many rows can be put in the buffer. If the cursor is opened in any mode other than NO SCROLL, Sybase IQ allows for backward scrolling for up to the total number of rows allowed in the buffer before it must restart the query. This is not true for NO SCROLL cursors as they do not allow backward scrolling. For example, with the default value for this option, the buffer initially holds rows 1 through 200 of the query result set. If you fetch the first 300 rows, the buffer holds rows 101 through 300. You can scroll backward or forward within that buffer with very little overhead cost. If you scroll before row 101, Sybase IQ restarts that query until the desired row is back in the buffer. This can be an expensive operation to perform, so your application should avoid it where possible. An option is to increase the value for CURSOR_WINDOW_ROWS to accommodate a larger possible scrolling area; however, the default setting of 200 is sufficient for most applications.

DATE_FIRST_DAY_OF_WEEK option

Function: Determines the first day of the week.

Allowed values: 0 – 6

Default: 0 (Sunday)

Scope: DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description: This option can specify which day is the first day of the week. By default, Sunday is day 1, Monday is day 2, Tuesday is day 3, and so on. Table 2-9defines the valid values for the DATE_FIRST_DAY_OF_WEEK option.

Table 2-8: DATE_FIRST_DAY_OF_WEEK values

Value	First Day
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

For example, if you change the value for the DATE_FIRST_DAY_OF_WEEK option to 3, Wednesday becomes day 1, Thursday becomes day 2, and so on. This option only affects the DOW and DATEPART functions, so its effect is quite narrow.

See also: The Adaptive Server Anywhere option FIRST_DAY_OF_WEEK performs the same function but assigns the values 1 through 7 instead of 0 through 6. 1 stands for Monday and 7 for Sunday (the default). If you receive unexpected results, see Ordering query results in *Sybase IQ Performance and Tuning Guide*.

DATE_FORMAT option

Function: Sets the format used for dates retrieved from the database.

Allowed values: String

Default: 'YYYY-MM-DD'. This corresponds to ISO date format specifications.

Scope

Can be set for an individual connection or the PUBLIC group. Takes effect immediately.

Description

The format is a string using the following symbols:

Table 2-9: Symbols used in DATE_FORMAT string

Symbol	Description
yy	2-digit year
yyyy	4-digit year
mm	2-digit month, or 2-digit minutes if following a colon (as in 'hh:mm')
mmm	3-character name of month
mmmm[m...]	Character long form for months—as many characters as there are m's, until the number of m's specified exceeds the number of characters in the month's name.
d	Single-digit day of week, (0 = Sunday, 6 = Saturday)
dd	2-digit day of month
ddd	3-character name of the day of week.
dddd[d...]	Character long form for day of the week—as many characters as there are d's, until the number of d's specified exceeds the number of characters in the day's name.
hh	2-digit hours
nn	2-digit minutes
ss[.s...s]	Seconds and parts of a second; up to six digits can follow the decimal point
aa	AM or PM (12 hour clock)
pp	PM if needed (12 hour clock)
jjj	Day of the year, from 1 to 366

Note Multibyte characters are not supported in date format strings. Only single-byte characters are allowed, even when the collation order of the database is a multibyte collation order like 932JPN. Use the concatenation operator to include multibyte characters in date format strings. For example, if '?' represents a multibyte character, use the concatenation operator to move the multibyte character outside of the date format string:

```
SELECT DATEFORMAT (start_date, 'yy') + '?'
FROM employee;
```

Each symbol is substituted with the appropriate data for the date being formatted. Any format symbol that represents character rather than digit output can be put in uppercase which causes the substituted characters to also be in uppercase. For numbers, using mixed case in the format string suppresses leading zeros.

You can control the padding of numbers by changing the case of the symbols. Same-case values (MM, mm, DD, or dd) all pad number with zeros. Mixed-case (Mm, mM, Dd, or dD) cause the number to not be zero-padded; the value takes as much room as required. For example:

```
SELECT dateformat ( cast ('1998/01/01' as date ), 'yyyy/
Mm/Dd' )
```

returns the following value:

```
1998/1/1
```

Examples

Table 2-10 illustrates DATE_FORMAT settings, together with the output from the following statement, executed on Thursday May 21, 1998:

```
SELECT CURRENT DATE
```

Table 2-10: DATE_FORMAT settings

DATE_FORMAT	SELECT CURRENT DATE
yyyy/mm/dd/ddd	1998/05/21/thu
jjj	141
mmm yyyy	may 1998
mm-yyyy	05-1998

See also

"Setting options" on page 24.

"RETURN_DATE_TIME_AS_STRING option" on page 143.

"TIME_FORMAT option" on page 165.

DATE_ORDER option

Function

Controls the interpretation of date formats.

Allowed values

'MDY', 'YMD', or 'DMY'

Default

'YMD'. This corresponds to ISO date format specifications.

Description

The database option DATE_ORDER is used to determine whether 10/11/12 is Oct 11 1912, Nov 12 1910, or Nov 10 1912. The option can have the value 'MDY', 'YMD', or 'DMY'.

DBCC_LOG_PROGRESS option

Function — Reports the progress of the sp_iqcheckdb system stored procedure.

Allowed values — ON, OFF

Default — OFF

Scope — Can be set for an individual connection or the PUBLIC group. Takes effect at the next execution of sp_iqcheckdb.

Description — When the DBCC_LOG_PROGRESS option is ON, the sp_iqcheckdb system stored procedure sends progress messages to the IQ message file. These messages allow the user to follow the progress of the sp_iqcheckdb operation.

Examples — The following is sample progress log output of the command `sp_iqcheckdb 'check database'`

```
IQ Utility Check Database
Start CHECK STATISTICS table: tloansf
Start CHECK STATISTICS for field: aqsn_dt
Start CHECK STATISTICS processing index:
ASIQ_IDX_T444_C1_FP
Start CHECK STATISTICS processing index:
tloansf_aqsn_dt_HNG
Done CHECK STATISTICS field: aqsn_dt
```

The following is sample progress log output of the command `sp_iqcheckdb 'allocation table nation'`

```
Start ALLOCATION table: nation
Start ALLOCATION processing index: nationhg1
Done ALLOCATION table: nation
Done ALLCOATION processing index: nationhg1
```

See also — Chapter 2, “System Recovery and Database Repair” in the *Sybase IQ Troubleshooting and Recovery Guide*

“sp_iqcheckdb procedure” on page 743.

DBCC_PINNABLE_CACHE_PERCENT option

Function — Controls the percent of the cache used by the sp_iqcheckdb system stored procedure.

Allowed values — 0 – 100

Default — 50

Scope — Can be set for an individual connection or the PUBLIC group. Takes effect at the next execution of sp_iqcheckdb.

Description — The sp_iqcheckdb system stored procedure works with a fixed number of buffers, as determined by this option. By Default, a large percentage of the cache is reserved to maximize sp_iqcheckdb performance.

See also — "sp_iqcheckdb procedure" on page 743.

Resource issues running sp_iqcheckdb in Chapter 2, "System Recovery and Database Repair"in the *Sybase IQ Troubleshooting and Recovery Guide*.

Chapter 2, "System Recovery and Database Repair" in the *Sybase IQ Troubleshooting and Recovery Guide*.

DDL_OPTIONS2 option

Function — Displays information about how long it took to insert or delete rows from an index.

Allowed values — 0 – 3

Default — 0

Scope — Can be set for an individual connection or the PUBLIC group. Takes effect immediately

Description — DDL_OPTIONS2 displays messages telling how long it took to insert or delete rows in each index. The following settings are allowed:

- 0 – Suppress messages
- 1 – Show messages on insert operations, such as LOAD TABLE
- 2 – Show messages on delete operations
- 3 – Show messages on insert and delete operations

For example, the following command displays messages that show how long it took to insert rows in each index:

```
SET TEMPORARY OPTION DDL_OPTIONS2 = 1
2004-04-06 09:24:10 0000000002 [20902]: Insert
completed. Index 'yahoo.DBA.ASIQ_IDX_T200_C10_FP', in 0
seconds.
```

See also — Interpreting notification messages in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

DEBUG_MESSAGES option

Function — Controls whether or not MESSAGE statements that include a DEBUG ONLY clause are executed.

Allowed values — ON, OFF

Default — OFF

Description — This option allows you to control the behavior of debugging messages in stored procedures that contain a MESSAGE statement with the DEBUG ONLY clause specified. By default, this option is set to OFF and debugging messages do not appear when the MESSAGE statement is executed. By setting DEBUG_MESSAGES to ON, you can enable the debugging messages in all stored procedures.

Note
DEBUG ONLY messages are inexpensive when the DEBUG_MESSAGES option is set to OFF, so these statements can usually be left in stored procedures on a production system. However, they should be used sparingly in locations where they would be executed frequently; otherwise, they might result in a small performance penalty.

See also — MESSAGE statement on page 600.

DEDICATED_TASK option

Function — Dedicates a request handling task to handling requests from a single connection.

Allowed values — ON, OFF

Default — OFF

Scope — Can be set as a temporary option only, for the duration of the current connection. Requires DBA permissions to set this option.

Description — When the DEDICATED_TASK connection option is set to ON, a request handling task is dedicated exclusively to handling requests for the connection. By pre-establishing a connection with this option enabled, you can gather information about the state of the database server if it becomes otherwise unresponsive.

DEFAULT_HAVING_SELECTIVITY option

Function	Provides default selectivity estimates to the optimizer for most HAVING clauses.
Allowed values	0 – 100
Default	0
Scope	Can be set for an individual connection or the PUBLIC group. Takes effect immediately.
Description	DEFAULT_HAVING_SELECTIVITY sets the selectivity for HAVING clauses, overriding optimizer estimates. A HAVING clause filters the results of a GROUP BY clause or a query with a select list consisting solely of aggregate functions. When DEFAULT_HAVING_SELECTIVITY is set to the default of 0, the optimizer estimates how many rows are filtered by the HAVING clause. Sometimes the IQ optimizer does not have sufficient information to choose an accurate selectivity, and in these cases chooses a generic estimate of 40%. DEFAULT_HAVING_SELECTIVITY allows a user to replace the optimizer estimate for all HAVING predicates in a query. Users can also specify the selectivity of individual HAVING clauses in the query, as described in the section "User-supplied condition hints" on page 199 in the "Search conditions" section, Chapter 3, "SQL Language Elements."
See also	Chapter 3, "Optimizing Queries and Deletions" in the *Sybase IQ Performance and Tuning Guide*.

DEFAULT_ISQL_ENCODING option [DBISQL]

Function	Specifies the code page that should be used by READ and OUTPUT statements.
Allowed values	*identifier* or *string*
Default	Use system code page (empty string)
Scope	Can be set as a temporary option only, for the duration of the current connection.
Description	DEFAULT_ISQL_ENCODING option is used to specify the code page to use when reading or writing files. It cannot be set permanently. The default code page is the default code page for the platform you are running on. On English Windows machines, the default code page is 1252.

Interactive SQL determines the code page that is used for a particular OUTPUT or READ statement as follows, where code page values occurring earlier in the list take precedence over those occurring later in the list:

- The code page specified in the ENCODING clause of the OUTPUT or READ statement
- The code page specified with the DEFAULT_ISQL_ENCODING option (if this option is set)
- The code page specified with the -codepage command line option when Interactive SQL was started
- The default code page for the computer Interactive SQL is running on

For a complete list of supported code pages, see the *Adaptive Server Anywhere Database Administration Guide*.

Example

Set the encoding to UTF-16 (for reading Unicode files):

```
SET TEMPORARY OPTION DEFAULT_ISQL_ENCODING = 'UTF-16'
```

See also

READ statement [DBISQL] and OUTPUT statement [DBISQL] in Chapter 6, "SQL Statements" in the *Sybase IQ Reference Manual*.

Pieces in the character set puzzle in Chapter 11, "International Languages and Character Sets" in the *Sybase IQ System Administration Guide*.

DEFAULT_LIKE_MATCH_SELECTIVITY option

Function

Provides default selectivity estimates to the optimizer for most LIKE predicates.

Allowed values

0 to 100

Default

15

Scope

Can be set for an individual connection or the PUBLIC group. Takes effect immediately.

Description

DEFAULT_LIKE_MATCH_SELECTIVITY sets the default selectivity for generic LIKE predicates, for example, `LIKE '`***`string%string`***`'` where % is a wildcard character. The optimizer relies on this option when other selectivity information is not available and the match string does not start with a set of constant characters followed by a single wildcard.

If the column has either a LF index or a 1- or 2-byte FP index, the optimizer can get exact information and does not need to use this value.

Users can also specify selectivity in the query, as described in the section "User-supplied condition hints" on page 199 in Chapter 3, "SQL Language Elements."

See also

"DEFAULT_LIKE_RANGE_SELECTIVITY option" on page 71.

"LIKE conditions" on page 193.

Chapter 3, "Optimizing Queries and Deletions" in the *Sybase IQ Performance and Tuning Guide*

DEFAULT_LIKE_RANGE_SELECTIVITY option

Function

Provides default selectivity estimates to the optimizer for leading constant LIKE predicates.

Allowed values

0 to 100

Default

15

Scope

Can be set for an individual connection or the PUBLIC group. Takes effect immediately.

Description

DEFAULT_LIKE_RANGE_SELECTIVITY sets the default selectivity for LIKE predicates, of the form `LIKE 'string%'` where the match string is a set of constant characters followed by a single wildcard character (%). The optimizer relies on this option when other selectivity information is not available.

If the column has either a LF index or a 1- or 2-byte FP index, the optimizer can get exact information and does not need to use this value.

Users can also specify selectivity in the query, as described in "User-supplied condition hints" on page 199 in the *Sybase IQ Reference Manual*.

See also

"DEFAULT_LIKE_MATCH_SELECTIVITY option" on page 70

"LIKE conditions" on page 193.

Chapter 3, "Optimizing Queries and Deletions" in the *Sybase IQ Performance and Tuning Guide*

DELAYED_COMMIT_TIMEOUT option

Function	Determines when the server returns control to an application following a COMMIT.
Allowed values	Integer, in milliseconds.
Default	500
Description	This option is ignored by Sybase IQ since DELAYED_COMMITS can only be set OFF.

DELAYED_COMMITS option

Function	Determines when the server returns control to an application following a COMMIT.
Allowed values	OFF
Default	OFF. This corresponds to ISO COMMIT behavior.
Description	When set to OFF (the only value allowed by Sybase IQ), the application must wait until the COMMIT is written to disk. This option must be set to OFF for ANSI/ISO COMMIT behavior.

DISABLE_RI_CHECK option

Function	Allows load, insert, update, or delete operations to bypass the referential integrity check, improving performance.
Allowed values	ON, OFF
Default	OFF
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	Users are responsible for ensuring that no referential integrity violation occurs during requests while DISABLE_RI_CHECK is set to ON.

DISK_STRIPING option

Function	Turns internal disk striping on or off.

Allowed values	ON, OFF
Default	ON
Scope	Can be set for the PUBLIC group only. Requires DBA permissions. Takes effect at the next checkpoint.
Description	This option can force disk striping to be set or unset for performance reasons. In Sybase IQ, disk striping places data in each dbspace file segment in a round-robin fashion; for example, the first database page written goes to the first dbspace, the second page written goes to the next dbspace, and so on). If the dbspace is in read-only or relocate mode, it is skipped. When this option is ON, the data is distributed in all dbspace segments, and the spaces tend to fill up together, evenly. When this option is OFF, the spaces tend to fill up in order, from the first to the last.
See also	For more information about disk striping and performance, see Balancing I/O in Chapter 5, "Managing System Resources" in the *Sybase IQ Performance and Tuning Guide*.

DISK_STRIPING_PACKED option

Function	Sets database block allocation policy.
Allowed values	ON, OFF
Default	ON
Scope	Can be set for the PUBLIC group only. Requires DBA permissions. Takes effect at the next checkpoint.
Description	In general, disk space fragmentation is much lower with the default setting, DISK_STRIPING_PACKED OFF. DISK_STRIPING_PACKED OFF causes Sybase IQ to check first for the first available space fragments to use when allocating space. Change this option only if Sybase Support instructs you to do so. When this option is ON, Sybase IQ allocates space starting from the next available last block allocated on the current dbspace.
See also	For more information about disk striping and performance, see Balancing I/O in Chapter 5, "Managing System Resources" of the *Sybase IQ Performance and Tuning Guide*.

DIVIDE_BY_ZERO_ERROR option [TSQL]

Function — Controls the reporting of division by zero.

Allowed values — ON, OFF

Default — ON

Description — This option indicates whether division by zero is reported as an error. If the option is set ON, then division by zero results in an error with SQLSTATE 22012.

If the option is set OFF, division by zero is not an error. Instead, a NULL is returned.

EARLY_PREDICATE_EXECUTION option

Function — Controls whether simple local predicates are executed before query optimization.

Allowed values — ON or OFF

Default — ON

Scope — DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description — If this option is ON (the default), the optimizer finds, prepares, and executes predicates containing only local columns and constraints before query optimization, including join ordering, join algorithm selection, and grouping algorithm selection, so that the values of "Estimated Result Rows" in the query plan are more precise. If this option is OFF, the optimizer finds and prepares the simple predicates, but does not execute them before query optimization. The resulting values of "Estimated Result Rows" are less precise, if the predicates are not executed.

In general, the EARLY_PREDICATE_EXECUTION option should always be left ON, as this results in improved query plans for many queries.

Note that when the EARLY_PREDICATE_EXECUTION option is ON, Sybase IQ executes the local predicates for all queries before generating a query plan, even when the NOEXEC option is ON. The generated query plan is the same as the runtime plan.

Query plan root node information – The following information is included in the query plan for the root node:

- Threads used for executing local invariant predicates: if greater than 1, indicates parallel execution of local invariant predicates
- Early_Predicate_Execution: indicates if the option is OFF
- Time of Cursor Creation: the time of cursor creation

Query plan leaf node information – The simple predicates whose execution is controlled by this option are referred to as invariant predicates in the query plan.The following information is included in the query plan for a leaf node, if there are any local invariant predicates on the node:

- Generated Post Invariant Predicate Rows: actual result after executing local invariant predicate
- Estimated Post Invariant Predicate Rows: calculated by using estimated local invariant predicates selectivity
- Time of Condition Start: starting time of the execution of local invariant predicates
- Time of Condition Done: ending time of the execution of local invariant predicates
- Elapsed Condition Time: elapsed time for executing local invariant predicates

ECHO option [DBISQL]

Function	Controls whether statements are echoed before they are executed.
Allowed values	ON, OFF
Default	ON
Description	This option is most useful when using the Windows READ statement to execute a DBISQL command file.

ENABLE_THREAD_ALLOWANCE option

Function	Allows optimizer to estimate thread allowance for each table that contains invariant predicates.
Allowed values	ON, OFF
Default	OFF

Scope — Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description — The IQ optimizer assigns a thread quota to each table that contains invariant predicates based on the row counts after high selectivity filters. This algorithm provides better thread allocation, which prevents the server from running out of threads.

ENABLED_ORDERED_PUSHDOWN_INSERTION option

Function — Controls how the query optimizer adds in the semijoin predicates for push-down joins selected by the join optimizer.

Allowed values — ON, OFF

Default — ON

Scope — Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description — Change this option only if Sybase Support instructs you to do so.

If OFF (the default), this option reverts the optimizer to the behavior seen in Sybase IQ 12.6.

If ON, this option allows the insertion of semijoin predicates in projection-savings order.

EXTENDED_JOIN_SYNTAX option

Function — Controls whether queries with an ambiguous syntax for multi-table joins are allowed, or reported as an error.

Allowed values — ON, OFF

Default — ON

Description — This option reports a syntax error for those queries containing outer joins that have ambiguous syntax due to the presence of duplicate correlation names on a null-supplying table.

The following join clause illustrates the kind of query that is reported where C1 is a condition.

```
( R left outer join T , T  join S on ( C1 ) )
```

If the EXTENDED_JOIN_SYNTAX option is set to ON, this query is interpreted as follows, where C1 and C2 are conditions:

```
( R left outer join T on ( C1 ) ) join S on ( C2 )
```

FLATTEN_SUBQUERIES option

Function	Enables the transformation of some simple correlated EXISTS and NOT EXISTS subqueries into equivalent join-based queries.
Allowed values	ON, OFF
Default	OFF
Scope	Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	Use only if instructed to do so by Sybase Technical Support. This option enables the transformation of some simple correlated EXISTS and NOT EXISTS subqueries into equivalent join-based queries. In most cases in which the optimizer can apply this transformation, the query runs faster with the transformation than without the transformation applied. There are some exceptions to this performance improvement and some queries might run more slowly, so be sure that use of this option is appropriate for your environment.

FLOAT_AS_DOUBLE option [TSQL]

Function	Controls the interpretation of the FLOAT keyword.
Allowed values	ON, OFF
Default	OFF
Description	Turning on the FLOAT_AS_DOUBLE option makes the IQ FLOAT keyword behave like the Adaptive Server Enterprise FLOAT keyword when a precision is not specified. When set to ON, Sybase IQ interprets all occurrences of the keyword FLOAT as equivalent to the keyword DOUBLE within SQL statements.

Note When using JDBC and Client Library connections, for example, running Sybase Central, you must set the FLOAT_AS_DOUBLE option to ON. If you do not do this, CREATE JOIN INDEX operations fails.

By default, IQ FLOAT values are interpreted by Sybase IQ as REAL values. Since Adaptive Server Enterprise treats its own FLOAT values as DOUBLE, enabling this option makes Sybase IQ to treat FLOAT values in the same way Adaptive Server Enterprise treats FLOAT values.

REAL values are four bytes, DOUBLE values are eight bytes. According to the ANSI SQL92 specification, FLOAT can be interpreted based on the platform. It is up to the database to decide what size it is, so long as it can handle the necessary precision. Adaptive Server Enterprise and Sybase IQ exhibit different default behavior.

Note If a join column is a REAL datatype, you must set FLOAT_AS_DOUBLE to OFF when creating join indexes, or an error occurs. Issues might also result when using inexact numerics for join columns.

The FLOAT_AS_DOUBLE option only takes effect when no precision is specified. For example the following statement is not affected by the option setting:

```
create table t1(
   c1 float(5)
)
```

The following statement is affected by the option setting:

```
create table t2(
   c1 float)
// affected by option setting
```

FORCE_DROP option

Function	Causes Sybase IQ to leak, rather than reclaim, database disk space during a DROP command.
Allowed values	ON, OFF
Default	OFF
Scope	Requires DBA permissions to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

You must drop a corrupt index, join index, column or table and set the FORCE_DROP option to ON. This prevents the free list from being incorrectly updated from incorrect or suspect file space allocation information in the object being dropped. After dropping corrupt objects, you can reclaim the file space using the -iqfrec and -iqdroplks server switches.

When force dropping objects, you must ensure that only the DBA is connected to the database. The server must be restarted immediately after a force drop.

If FORCE_DROP is set to ON, you cannot drop a join index on a multiplex write server. If you need to force drop a join index on a multiplex write server, you must first start the server in single-node mode. If FORCE_DROP = ON on the write server and an object is dropped, the FORCE_DROP option settings, on query servers are unaffected. After the drop restart only the write server.

Do not attempt to force drop objects unless Sybase Technical Support has instructed you to do so.

See also

For important information on using the FORCE_DROP option, see Chapter 2, "System Recovery and Database Repair" in the *Sybase IQ Troubleshooting and Recovery Guide*.

FORCE_NO_SCROLL_CURSORS option

Function

Forces all cursors to be non-scrolling.

Allowed values

ON, OFF

Default

OFF

Scope

DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

By default, all cursors are scrolling. Scrolling cursors with no host variable declared cause Sybase IQ to create a buffer for temporary storage of results. Each row in the result set is stored to allow for backward scrolling.

Setting FORCE_NO_SCROLL_CURSORS to ON forces all cursors to be non-scrolling, thereby saving on temporary storage requirements. This option can be useful if you are retrieving very large numbers (millions) of rows, however some front-end applications make use of scrolling cursor operations and require this option to be set OFF.

If scrolling cursors are never used in your application, you should make this a permanent public option. It uses less memory and makes a modest improvement in query performance.

FORCE_UPDATABLE_CURSORS option

Function	Controls whether cursors that have not been declared as updatable can be updated.
Allowed values	ON, OFF
Default	OFF
Scope	Can be set temporary, for an individual connection, for a group, or PUBLIC. Does not require DBA permissions. Takes effect immediately.
Description	When the FORCE_UPDATABLE_CURSORS option is ON, cursors which have not been declared as updatable can be updated. This option allows updatable cursors to be used in front-end applications without specifying the FOR UPDATE clause of the DECLARE CURSOR statement. Sybase does not recommend the use of the FORCE_UPDATABLE_CURSORS option unless absolutely necessary.

FPL_EXPRESSION_MEMORY_KB option

Function	Controls the use of memory for the optimization of queries involving functional expressions against columns having enumerated storage.
Allowed values	0 – 20000
Default	1024 kilobytes
Scope	Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	FPL_EXPRESSION_MEMORY_KB option controls the use of memory for the optimization of queries involving functional expressions against columns having enumerated storage. The option enables the DBA to constrain the memory used by this optimization and balance it with other Sybase IQ memory requirements, such as caches and LOAD_MEMORY_MB. Setting this option to 0 switches off optimization.

FP_PREDICATE_WORKUNIT_PAGES option

Function	Specifies degree of parallelism used in the default index.
Allowed values	Integer
Default	400

Scope — DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description — The default index calculates some predicates such as SUM, RANGE, MIN, MAX and COUNT DISTINCT in parallel. FP_PREDICATE_WORKUNIT_PAGES affects the degree of parallelism used by specifying the number of pages worked on by each thread. To increase the degree of parallelism, decrease the value of this option.

GARRAY_FILL_FACTOR_PERCENT option

Function — Specifies the amount of space to reserve for an HG index.

Allowed values — 0 – 1000

Default — 25

Scope — DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description — An HG index can reserve some storage on a per-group basis (where group is defined as a group of rows with identical values). Reserving space consumes some disk space but can help the performance of incremental inserts into the HG index and reduce fragmentation.

If you plan to do future incremental inserts into an HG index, and those new rows have values that are already present in the index, a nonzero value for this option helps.

GARRAY_INSERT_PREFETCH_SIZE option

Function — Specifies number of pages used for prefetch.

Allowed values — 0 – 100

Default — 3

Scope — DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description — This option defines the number of database pages read ahead during an insert to a column that has an HG index.

Do not set this option unless advised to do so by Sybase Technical Support.

GARRAY_RO_PREFETCH_SIZE option

Function	Specifies number of pages used for prefetch.
Allowed values	0 – 100
Default	10
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	This option defines the number of database pages read ahead during a query to a column that has an HG index. Do not set this option unless advised to do so by Sybase Technical Support.

HASH_PINNABLE_CACHE_PERCENT option

Function	Maximum percentage of a user's temp memory that a hash object can pin.
Allowed values	0 – 100
Default	20
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	HASH_PINNABLE_CACHE_PERCENT controls the percentage of a user's temp memory allocation that any one hash object can pin in memory. It defaults to 20%, but reduce this number to 10% for sites that run complex queries, or increase to 50% for sites with simple queries that need a single large hash object to run, such as a large IN subquery. The HASH_PINNABLE_CACHE_PERCENToption is for use by primarily Sybase Technical Support. If you change the value of it, do so with extreme caution; first analyze the effect on a wide variety of queries.
See also	"BIT_VECTOR_PINNABLE_CACHE_PERCENT option" on page 47. "SORT_PINNABLE_CACHE_PERCENT option" on page 146.

HASH_THRASHING_PERCENT option

Function	Specifies the percent of hard disk I/Os allowed during the execution of a statement that includes a query involving hash algorithms, before the statement is rolled back and an error message is reported.

Allowed values	0 – 100
Default	10
Scope	Can be set for an individual connection or the PUBLIC group. Takes effect immediately.
Description	If a query that uses hash algorithms causes an excessive number of hard disk I/Os (paging buffers from memory to disk), query performance is negatively affected, and server performance might also be affected. The HASH_THRASHING_PERCENT option controls the percentage of hard disk I/Os allowed before the statement is rolled back and an error message is returned. The text of the error message is either "Hash insert thrashing detected" or "Hash find thrashing detected." The default value of HASH_THRASHING_PERCENT is 10%. Increasing it permits more paging to disk before a rollback and decreasing it permits less paging before a rollback.
See also	For more information on controlling excessive paging and using the HASH_THRASHING_PERCENT option, see Unexpectedly long loads or queries in Chapter 1, "Troubleshooting Hints," in the *Sybase IQ Troubleshooting and Recovery Guide*. Also see "HASH_PINNABLE_CACHE_PERCENT option" on page 82.

HEADINGS option [DBISQL]

Function	Controls whether headings display for the results of a SELECT statement.
Allowed values	ON, OFF
Default	ON
Description	Set this option according to your preference.

HG_DELETE_METHOD option

Function	Specifies the algorithm used during a delete in a HG index.
Allowed values	0 – 3
Default	0
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

This option chooses the algorithm used by the HG index during a delete operation. The cost model considers the CPU related costs as well as I/O related costs in selecting the appropriate delete algorithm. The cost model takes into account:

- Rows deleted
- Index size
- Width of index data type
- Cardinality of index data
- Available temporary cache
- Machine related I/O and CPU characteristics
- Available CPUs and threads
- Referential integrity costs

To force a "small" method, set this option to 1. To force the "large" method, set the option to 2. To force a "midsize" method, set the option to 3.

See also

For more details about these methods, see Optimizing delete operations in *Sybase IQ Performance and Tuning Guide*.

HG_SEARCH_RANGE option

Function

Specifies the maximum number of Btree pages used in evaluating a range predicate in the HG index.

Allowed values

Integer

Default

10

Scope

Can be set for an individual connection or the PUBLIC group. Takes effect immediately.

Description

The default setting of this option is appropriate for most queries.

IDENTITY_ENFORCE_UNIQUENESS option

Function

Creates a unique HG index on each Identity/Autoincrement column if the column is not already a primary key.

Allowed values

ON, OFF

Default	OFF
Scope	Can only be set temporary (for a connection), for a user, or for the PUBLIC group. Takes effect immediately.
Description	When option is set ON, HG indexes are created on future identity columns. The index can only be deleted if the deleting user is the only one using the table and the table is not a local temporary table.
See also	"QUERY_PLAN option" on page 139.

IDENTITY_INSERT option

Function	Enables users to insert values into or to update an IDENTITY or AUTOINCREMENT column.
Allowed values	= '*tablename*'
Default	Option not set.
Scope	Can be set only temporary (for a connection), for a user, or for the PUBLIC group. Takes effect immediately.
Description	When option is set, insert/update is enabled. A table name must be specified to identify the column to insert or update. If you are not the table owner, qualify the table name with the owner name. To drop a table with an IDENTITY column, IDENTITY_INSERT must not be set to that table.

Example	For example, if you use the table employees to run explicit inserts:
	
	`SET TEMPORARY OPTION IDENTITY_INSERT = 'employees'`
	
	To turn the option off, specify the equals sign and an empty string:
	
	`SET TEMPORARY OPTION IDENTITY_INSERT = ''`

See also	"QUERY_PLAN option" on page 139

INDEX_ADVISOR option

Function	Generates messages suggesting additional column indexes that may improve performance of one or more queries.
Allowed values	ON, OFF

Default: OFF

Scope: Can be set temporary (for a connection), for a user, or for the PUBLIC group. Takes effect immediately.

Description: When set ON, the index advisor prints index recommendations as part of the Sybase IQ query plan or as a separate message in the Sybase IQ message log file if query plans are not enabled. These messages begin with the string "Index Advisor:" and you can use that string to search and filter them from a Sybase IQ message file. The output is in OWNER.TABLE.COLUMN format.

Set both INDEX_ADVISOR and INDEX_ADVISOR_MAX_ROWS to accumulate index advice.

Note When INDEX_ADVISOR_MAX_ROWS is set ON, index advice will not be written to the Sybase IQ message file as separate messages. Advice will, however, continue to be displayed on query plans in the Sybase IQ message file.

Table 2-11: Index Advisor

Situation	Recommendation
Local predicates on a single column where an HG, LF, HNG, DATE, TIME or DATETIME index would be desirable, as appropriate.	Recommend adding an <index-type> index to column <col>
Single column join keys where an LF or HG index would be useful.	Add an LF or HG index to join key <col>
Single column candidate key indexes where a HG exists, but could be changed to a unique HG or LF	Change join key <col> to a unique LF or HG index
Join keys have mismatched data types, and regenerating one column with a matched data type would be beneficial.	Make join keys <col1> and <col2> identical data types
Subquery predicate columns where an LF or HG index would be useful.	Add an LF or HG index to subquery column <col>
Grouping columns where an LF or HG index would be useful.	Create an LF or HG index on grouping column <col>
Single-table intercolumn comparisons where the two columns are identical data types, a CMP index are recommended.	Create a CMP index on <col1>, <col2>
Columns where an LF or HG index exists, and the number of distinct values allows, suggest converting the FP to a 1 or 2-byte FP index.	Rebuild <col> with 'optimize storage=on'

It is up to you to decide how many queries benefit from the additional index and whether it is worth the expense to create and maintain the indexes. In some cases, you cannot determine how much, if any, performance improvement results from adding the recommended index.

For example, consider columns used as a join key. Sybase IQ uses metadata provided by HG or LF indexes extensively to generate better/faster query plans to execute the query. Putting an HG or LF index on a join column without one makes the IQ optimizer far more likely to choose a faster join plan, but without adding the index and running the query again, it is very hard to determine whether query performance stays the same or improves with the new index.

Example

Index advisor output with query plan set OFF.

```
I. 03/30 14:18:45. 0000000002 Advice: Add HG or LF index
on DBA.ta.c1  Predicate: (ta2.c1 < BV(1))
```

Index advisor output with query plan set ON.

Note This method accumulates index advisor information for multiple queries so that advice for several queries can be tracked over time in a central location.

```
I. 03/30 14:53:24. 0000000008 [20535]: 6   .....#03:
Leaf
I. 03/30 14:53:24. 0000000008
[20535]:               Table    Name: tb
I. 03/30 14:53:24. 0000000008
[20535]:               Condition 1 (Invariant): (tb.c3
=tb.c4)
I. 03/30 14:53:24. 0000000008
[20535]:               Condition 1 Index Advisor: Add a
  CMP index on DBA.tb (c3,c4)
```

See also

"QUERY_PLAN option" on page 139

Message logging in Chapter 1, "Overview of Sybase IQ System Administration" in the *Sybase IQ System Administration Guide*

"INDEX_ADVISOR_MAX_ROWS option" on page 87

"sp_iqindexadvice procedure" on page 788

INDEX_ADVISOR_MAX_ROWS option

Function

Sets the maximum number of unique advice messages stored to max_rows.

Allowed values

Value	Description
0	Minimum value disables collection of index advice
4294967295	Maximum value allowed

Default	0
Scope	Can be set temporary (for the current connection), or persistent for a user/group (such as PUBLIC or DBA). Takes effect immediately.
Description	The INDEX_ADVISOR_MAX_ROWS option is used to limit the number of messages stored by the index advisor. Once the specified limit has been reached, the INDEX_ADVISOR will not store new advice. It will, however, continue to update counts and timestamps for existing advice messages. `SET OPTION public.Index_Advisor_Max_Rows = max_rows;`
See also	"INDEX_ADVISOR option" on page 85 "sp_iqindexadvice procedure" on page 788

INDEX_PREFERENCE option

Function	Controls the choice of indexes to use for queries.
Allowed values	-10 to 10
Default	0
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	The Sybase IQ optimizer normally chooses the best index available to process local WHERE clause predicates and other operations that can be done within an IQ index. INDEX_PREFERENCE is used to override the optimizer choice for testing purposes; under most circumstances, it should not be changed. Table 2-12 describes the valid values for this option and their action.

Table 2-12: INDEX_PREFERENCE values

Value	Action
0	Let the optimizer choose
1	Prefer LF indexes
2	Prefer HG indexes
3	Prefer HNG indexes
4	Prefer CMP indexes
5	Prefer the default index
6	Prefer WD indexes
8	Prefer DATE indexes
9	Prefer TIME indexes
10	Prefer DTTM indexes
-1	Avoid LF indexes
-2	Avoid HG indexes
-3	Avoid HNG indexes
-4	Avoid CMP indexes
-5	Avoid the default index
-6	Avoid WD indexes
-8	Avoid DATE indexes
-9	Avoid TIME indexes
-10	Avoid DTTM indexes

INFER_SUBQUERY_PREDICATES option

Function Controls the optimizer's inference of additional subquery predicates.

Allowed values ON, OFF

Default OFF

Scope Can be set temporary for an individual connection or the PUBLIC group. Takes effect immediately. DBA permissions are not required to set this option.

Description INFER_SUBQUERY_PREDICATES controls whether the optimizer is allowed to infer additional subquery predicates from an existing subquery predicate through transitive closure across a simple equality join predicate. In most cases in which the optimizer chooses to make this inference, the query runs faster. There are some exceptions to this performance improvement, so you may need to experiment to be sure that this option is appropriate for your environment.

IN_SUBQUERY_PREFERENCE option

Function — Controls the choice of algorithms for processing an IN subquery.

Allowed values — -3 to 3

Default — 0

Scope — DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description — The IQ optimizer has a choice of several algorithms for processing IN subqueries. This option allows you to override the optimizer's costing decision when choosing the algorithm to use. It does not override internal rules that determine whether an algorithm is legal within the query engine.

IN_SUBQUERY_PREFERENCE is normally used for internal testing and for manually tuning queries that the optimizer does not handle well. Only experienced DBAs should use it. The only reason to use this option is if the optimizer seriously underestimates the number of rows produced by a subquery, and the hash object is thrashing. Before setting this option, try to improve the mistaken estimate by looking for missing indexes and dependent predicates.

Inform Sybase Technical Support if you need to set IN_SUBQUERY_PREFERENCE, as setting this option might mean that a change to the optimizer is appropriate.

Table 2-13 describes the valid values for this option and their actions.

Table 2-13: IN_SUBQUERY_PREFERENCE values

Value	Action
0	Let the optimizer choose
1	Prefer sort-based IN subquery
2	Prefer vertical IN subquery (where a subquery is a child of a leaf node in the query plan)
3	Prefer hash-based IN subquery
-1	Avoid sort-based IN subquery
-2	Avoid vertical IN subquery
-3	Avoid hash-based IN subquery

IQGOVERN_MAX_PRIORITY option

Function	Limits the allowed IQGOVERN_PRIORITY setting.
Allowed values	1 – 3
Default	2
Scope	Can be set temporary, per user, or PUBLIC. Requires DBA permissions to set. Takes effect immediately.
Description	Limits the allowed IQGOVERN_PRIORITY setting, which affects the order in which a user's queries are queued for execution. In the range of allowed values, 1 indicates high priority, 2 (the default) medium priority, and 3 low priority. Sybase IQ returns an error if a user sets IQGOVERN_PRIORITY higher than IQGOVERN_MAX_PRIORITY.

IQGOVERN_PRIORITY option

Function	Assigns a priority to each query waiting in the -iqgovern queue.
Allowed values	1 – 3
Default	2
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	Assigns a value that determines the order in which a user's queries are queued for execution. In the range of allowed values, 1 indicates high priority, 2 (the default) medium priority, and 3 low priority. This switch can be set temporary, per user, or public by any user. Queries with a lower priority will not run until all higher priority queries have executed. This option is limited by the per user or per group value of the option IQGOVERN_MAX_PRIORITY.

IQGOVERN_PRIORITY_TIME option

Function	Limits the time a high priority query waits in the queue before starting.
Allowed values	1 – 1,000,000 seconds. Must be lower than IQGOVERN_MAX_PRIORITY.
Default	0 (disabled)

Scope — Can be set for the PUBLIC group only. Requires DBA permissions. Takes effect immediately.

Description — Limits the time a high priority (priority 1) query waits in the queue before starting. When the limit is reached, the query is started even if it exceeds the number of queries allowed by the -iqgovern setting. You must belong to group DBA in order to change this switch. The range is from 1 to 1,000,000 seconds. The default (0) disables this feature.

IQMSG_LENGTH_MB option

Function — Enables IQ message log file wrapping, and sets maximum size of this file.

Allowed values — Integer between 0 and 2047, in megabytes

Default — 0 (message log wrapping disabled)

Scope — Can be set only for the PUBLIC group. Requires DBA privileges to set. Takes effect immediately.

Description — When you start a database, messages are recorded in the *.iqmsg* log file, and each new line is appended to the end of the file.

Initially, message log wrapping is disabled (IQMSG_LENGTH_MB is 0), messages are always appended to the end of the file, and the file continues to grow.

When message log wrapping is enabled (IQMSG_LENGTH_MB is greater than 0), the *.iqmsg* log file can grow only to the specified size. The maximum size you can specify is 2047 (2GB). When it reaches that size, new messages are written to the beginning of the file, overwriting existing messages on a line-by-line basis.

If wrapping is enabled when the database is shut down, it is still enabled the next time the database is started. If you then disable wrapping, by setting IQMSG_LENGTH_MB = 0, Sybase IQ writes new messages continuously to the ending position of the most recent message, overwriting any existing messages, until it reaches the end of the file. From then on, it appends new messages to the end of the file.

When wrapping is enabled, three tags remind you that the last message in the file might not be the most recent message, and help you identify where new messages are being placed.

- This tag indicates the ending position of the most recent message:

```
<next msg insertion place>
```

- This tag occurs at the start of the file:

```
!!!!!! log wrapped back here from the end of the file
!!!!!!
```

- This tag occurs at the end of the file:

```
!!!!!! log wrapped back to the beginning of the file
!!!!!!
```

If a database file already exists and its *.iqmsg* file is larger than IQMSG_LENGTH_MB, the maximum file size is the actual file size. Setting this option does not truncate the file.

ISOLATION_LEVEL option

Function	Controls the locking isolation level for Catalog Store tables.
Allowed values	0, 1, 2, or 3
Default	0
Description	Each locking isolation level is defined as follows:

- 0 – Allow dirty reads, nonrepeatable reads, and phantom rows.
- 1 – Prevent dirty reads. Allow nonrepeatable reads and phantom rows.
- 2 – Prevent dirty reads and guarantee repeatable reads. Allow phantom rows.
- 3 – Serializable. Do not allow dirty reads, guarantee repeatable reads, and do not allow phantom rows.

ISOLATION_LEVEL determines the isolation level for tables in the Catalog Store. Sybase IQ always enforces level 3 for tables in the IQ Store. Level 3 is equivalent to ANSI level 4.

ISQL_COMMAND_TIMING option [DBISQL]

Function	Controls whether SQL statements are timed or not.
Allowed values	ON, OFF
Default	ON

Description

This boolean option controls whether SQL statements are timed or not. If you set the option to ON, the time of execution appears in the Messages pane after you execute a statement. If you set the option to OFF, the time does not appear.

You can also set this option on the Messages tab of the Options dialog.

ISQL_ESCAPE_CHARACTER option [DBISQL]

Function

Controls the escape character used in place of unprintable characters in data exported to ASCII files.

Allowed values

Any single character

Default

A backslash (\ )

Description

When Interactive SQL exports strings that contain unprintable characters (such as a carriage return), it converts each unprintable character into a hexadecimal format and precedes it with an escape character. The character you specify for this setting is used in the output if your OUTPUT statement does not contain an ESCAPE CHARACTER clause. This setting is used only if you are exporting to an ASCII file.

Example

- Create a table that contains one string value with an embedded carriage return (denoted by the "\n" in the INSERT statement). Then export the data to *c:\escape.txt* with a # sign as the escape character.

```
CREATE TABLE escape_test( TEXT varchar(10 ) );
INSERT INTO escape_test VALUES( 'one\ntwo' );
SET TEMPORARY OPTION ISQL_ESCAPE_CHARACTER='#';
SELECT * FROM escape_test;
OUTPUT TO c:\escape.txt FORMAT ASCII
```

  This code places the following data in *escape.txt*:

  'one#x0Atwo'

  where # is the escape character and *x0A* is the hexadecimal equivalent of the "\n" character.

  The start and end characters (in this case, single quotation marks) depend on the ISQL_QUOTE setting.

ISQL_FIELD_SEPARATOR option [DBISQL]

Function
Controls the default string used for separating values in data exported to ASCII files.

Allowed values
String

Default
A comma (,)

Description
Controls the default string used for separating (or delimiting) values in data exported to ASCII files. If an OUTPUT statement does not contain a DELIMITED BY clause, the value of this setting is used.

Example
- Set the field separator to a colon in the data exported to *c:\employee.txt*.

```
SET TEMPORARY OPTION ISQL_FIELD_SEPARATOR=':';
SELECT emp_lname, emp_fname FROM employee
WHERE emp_id < 150;
OUTPUT TO c:\employee.txt FORMAT ASCII
```

This code places the following data in *employee.txt*:

'Whitney':'Fran'

'Cobb':'Matthew'

'Chin':'Philip'

'Jordan':'Julie'

The start and end characters (in this case, single quotation marks) depend on the ISQL_QUOTE setting.

ISQL_LOG option [DBISQL]

Function
Controls logging behavior.

Allowed values
String containing a file name

Default
" (the empty string)

Description
If ISQL_LOG is set to a nonempty string, all Interactive SQL statements are added to the end of the named file. Otherwise, if ISQL_LOG is set to the empty string, Interactive SQL statements are not logged.

This option logs an individual Interactive SQL session only.

ISQL_QUOTE option [Interactive SQL]

Function
Controls the default string that begins and ends all strings in data exported to ASCII files.

Allowed values
String

Default
A single apostrophe (')

Description
Controls the default string that begins and ends all strings in data exported to ASCII files. If an OUTPUT statement does not contain a QUOTE clause, this value is used by default.

Example

- To change the default string that begins and ends all strings to a double quote character.

```
SET TEMPORARY OPTION ISQL_QUOTE='"'; SELECT
emp_lname, emp_fname FROM employee WHERE emp_id <
150; OUTPUT TO c:\employee.txt FORMAT ASCII
```

  This code places the following data in *employee.txt*:

  “Whitney”, “Fran”

  “Cobb”,”Matthew”

  “Chin”,”Philip”

  “Jordan”,”Julie”

  The separator characters (in this case, commas) depend on the ISQL_FIELD_SEPARATOR setting.

JAVA_HEAP_SIZE option

Function
Limits the memory used by Java applications for a connection.

Allowed values
Integer

Default
1000000

Scope
Can be set for an individual connection or the PUBLIC group. Takes effect immediately. Requires DBA permissions to set this option for *any* connection.

Description
This option sets the maximum size (in bytes) of that part of the memory that is allocated to Java applications on a per connection basis. Per-connection memory allocations typically consist of the user’s working set of allocated Java variables and Java application stack space.

While a Java application is executing on a connection, the per-connection allocations come out of the fixed cache of the database server, so it is important that a runaway Java application be disallowed from using up too much memory.

JAVA_NAMESPACE_SIZE option

Function — Limits the memory used by Java applications for a database.

Allowed values — Integer

Default — 4000000

Description — This option sets the maximum size (in bytes) of that part of the memory that is allocated to Java applications on a per-database basis.

Per-database memory allocations include Java class definitions. As class definitions are effectively read-only, they are shared between connections. Consequently, their allocations come directly from the fixed cache, and JAVA_NAMESPACE sets a limit on the size of these allocations.

JOIN_EXPANSION_FACTOR option

Function — Controls how conservative the optimizer's join result estimates are in unusually complex situations.

Allowed values — 1 – 100

Default — 30

Scope — Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description — This option controls how conservative the join optimizer's result size estimates are in situations where an input to a specific join has already passed through at least one intermediate join that can result in multiple copies of rows projected from the table being joined.

A level of zero indicates that the optimizer should use the same estimation method above intermediate expanding joins as it would if there were no intermediate expanding joins.

This results in the most aggressive (small) join result size estimates.

A level of 100 indicates that the optimizer should be much more conservative in its estimates whenever there are intermediate expanding joins, and this results in the most conservative (large) join result size estimates.

Normally, you should not need to change this value. If you do, Sybase recommends setting JOIN_EXPANSION_FACTOR as a temporary or user option.

JOIN_OPTIMIZATION option

Function

Enables or disables the optimization of the join order.

Allowed values

ON, OFF

Default

ON

Scope

DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

When the JOIN_OPTIMIZATION option is ON, Sybase IQ optimizes the join order to reduce the size of intermediate results and sorts, and to balance the system load. When the option is OFF, the join order is determined by the order of the tables in the FROM clause of the SELECT statement.

JOIN_OPTIMIZATION should always be set ON.

The JOIN_OPTIMIZATION option controls the order of the joins, but not the order of the tables. To show the distinction, consider this example FROM clause with four tables:

```
FROM  A, B, C, D
```

By default, this FROM clause creates a left deep plan of joins that could also be explicitly represented as:

```
FROM  (((A, B), C), D)
```

If JOIN_OPTIMIZATION is turned OFF, then the order of these joins on the sets of tables is kept precisely as specified in the FROM clause. Thus A and B must be joined first, then that result must be joined to table C, and then finally joined to table D. This option does not control the left/right orientation at each join. Even with JOIN_OPTIMIZATION turned OFF, the optimizer, when given the above FROM clause, can produce a join plan that looks like:

```
FROM  ((C, (A, B)), D)
```

or

```
FROM  (((B, A), C), D)
```

or

```
FROM  (D, ((A, B), C))
```

In all of these cases, A and B are joined first, then that result is joined to C, and finally that result is joined to table D. The order of the joins remains the same, but the order of the tables appears different.

In general, if JOIN_OPTIMIZATION is turned OFF, you probably should use parentheses in the FROM clause, as in the above examples, to make sure that you get the join order you want. If you want to join A and B to the join of C and D, you can specify this join by using parentheses:

```
FROM ((A, B), (C, D))
```

Note that the above FROM clause is a different join order than the original example FROM clause, even though all the tables appear in the same order.

JOIN_OPTIMIZATION should be set to OFF only to diagnose obscure join performance issues or to manually optimize a small number of predefined queries. With JOIN_OPTIMIZATION turned OFF, queries can join up to 128 tables, but might also suffer serious performance degradation.

Warning! If you turn off JOIN_OPTIMIZATION, Sybase IQ has no way to ensure optimal performance for queries containing joins. You assume full responsibility for performance aspects of your queries.

JOIN_PREFERENCE option

Function	Controls the choice of algorithms when processing joins.
Allowed values	-7 to 7
Default	0
Scope	DBA permissions are not required to set JOIN_PREFERENCE. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

For joins within a query, the IQ optimizer has a choice of several algorithms for processing the join. JOIN_PREFERENCE allows you to override the optimizer's cost-based decision when choosing the algorithm to use. It does not override internal rules that determine whether an algorithm is legal within the query engine. If you set it to any nonzero value, every join in a query is affected; you cannot use it to selectively modify one join out of several in a query.

This option is normally used for internal testing, and only experienced DBAs should use it. Table 2-14 describes the valid values for this option and their action.

Table 2-14: JOIN_PREFERENCE values

Value	Action
0	Let the optimizer choose
1	Prefer sort-merge
2	Prefer nested-loop
3	Prefer nested-loop push-down
4	Prefer hash
5	Prefer hash push-down
6	Prefer prejoin
7	Prefer sort-merge push-down
-1	Avoid sort-merge
-2	Avoid nested-loop
-3	Avoid nested-loop push-down
-4	Avoid hash
-5	Avoid hash push-down
-6	Avoid prejoin
-7	Avoid sort-merge push-down

JOIN_SIMPLIFICATION_THRESHOLD option

Function

Controls the minimum number of tables being joined together before any join optimizer simplifications are applied.

Allowed values

1 – 64

Default

15

Scope

Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

The query optimizer simplifies its optimization of join order by separate handling of both lookup tables (that is, nonselective dimension tables) and tables that are effective Cartesian products. After simplification, it optimizes the remaining tables for join order, up to the limit set by MAX_JOIN_ENUMERATION.

Setting this option to a value greater than the current value for MAX_JOIN_ENUMERATION has no effect.

Setting this value below the value for MAX_JOIN_ENUMERATION might improve the time required to optimize queries containing many joins, but may also prevent the optimizer from finding the best possible join plan.

Normally, you should not need to change this value. If you do, Sybase recommends setting JOIN_SIMPLIFICATION_THRESHOLD as a temporary or user option, and to a value of at least 9.

LARGE_DOUBLES_ACCUMULATOR option

Function

Controls which accumulator to use for SUM or AVG of floating-point numbers.

Allowed values

ON, OFF

Default

OFF

Scope

DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

The small accumulator for floats and doubles is highly accurate for addends in the range of magnitudes 1e-20 to 1e20. It loses some accuracy outside of this range but is still good enough for many applications. The small accumulator allows the optimizer to choose hash for faster performance more easily than the large accumulator. The large accumulator is highly accurate for all floats and doubles, but its size often precludes the use of hash optimization. The default is the small accumulator.

LF_BITMAP_CACHE_KB option

Function

Specifies the amount of memory to use for a load into a LF index.

Allowed values

1 – 8

Default

4

Scope

DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

LF_BITMAP_CACHE_KB defines the amount of heap memory (in KB) per distinct value used during a load into an LF index. The default allots 4KB. If the sum of the distinct counts for all LF indexes on a particular table is relatively high (greater than 10,000), then heap memory use might increase to the point of impacting load performance due to system page faulting. If this is the case, reduce the value of LF_BITMAP_CACHE_KB.

The following formula shows how to calculate the heap memory used (in bytes) by a particular LF index during a load:

```
Heap-memory-used = (lf_bitmap_cache_kb * 1024)
* lf-distinct-count-for-column
```

Using the default of 4KB, an LF index with 1000 distinct values can use up to 4MB of heap memory during a load.

LOAD_MEMORY_MB option

Function

Specifies an upper bound (in MB) on the amount of heap memory subsequent loads can use.

Allowed values

0 – 2000

Default

0 (zero)

Scope

DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

This option specifies an upper bound (in MB) on the amount of heap memory subsequent loads can use. The default setting, 0, means that there is no upper bound, and Sybase IQ can use as much heap memory as necessary to perform the load. A nonzero value means that the user has set an upper bound. The maximum upper bound is 2000MB (2GB). This option is typically used for LOAD statements, but affects all operations where loads, inserts, or updates occur, including SYNCHRONIZE, DELETE, INSERT and UPDATE operations.

LOAD_ZEROLENGTH_ASNULL option

Function
Specifies LOAD statement behavior under certain conditions.

Allowed values
ON, OFF

DBA permissions are not required to set LOAD_ZEROLENGTH_ASNULL. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Default
OFF

Description
This option specifies LOAD statement behavior under the following conditions:

- inserting a zero-length data value into a column of data type CHAR, VARCHAR, LONG VARCHAR, BINARY, VARBINARY, or LONG BINARY

  and

- a NULL column-spec; for example, NULL(ZEROS) or NULL(BLANKS) is also given for that same column

Set LOAD_ZEROLENGTH_ASNULL ON to load a zero-length value as NULL when the above conditions are met.

Set LOAD_ZEROLENGTH_ASNULL OFF to load a zero-length value as zero-length, subject to the setting of option NON_ANSI_NULL_VARCHAR.

See also
"NON_ANSI_NULL_VARCHAR option" on page 124

"LOAD TABLE statement" on page 580

LOCAL_KB_PER_STRIPE option

Function
Defines the number of kilobytes (KB) to write to each dbspace before the disk striping algorithm moves to the next stripe for the IQ Local Store.

Allowed values
Integer greater than zero, in kilobytes

Default
1 (which rounds up to one page)

Scope
Can be set only for the PUBLIC group. Requires DBA permissions to set the option. Takes effect at the next checkpoint.

Description

LOCAL_KB_PER_STRIPE lets you control the number of kilobytes written to each dbspace before the IQ disk striping algorithm moves to the next stripe for an IQ Local Store. The corresponding number of blocks is rounded up to a page boundary, so the actual amount written to each stripe might be slightly larger than requested. You can tune this option by measuring the time required to complete I/O intensive updates and adjusting the option value accordingly.

See also

Chapter 5, "Working with Database Objects" in the *Sybase IQ System Administration Guide*.

"Balancing I/O" on page 135 in Chapter 5, "Managing System Resources" in the *Sybase IQ Performance and Tuning Guide*.

LOCAL_RESERVED_DBSPACE_MB option

Function

Controls the amount of space Sybase IQ reserves in the IQ Local Store on a multiplex query server.

Allowed values

Integer greater than zero, in megabytes

Default

200; Sybase IQ actually reserves the minimum of 200MB or 50% of the size of the last dbspace

Scope

Can be set only for the PUBLIC group. DBA permissions are required to set the option. You must shut down and restart the database server for the change to take effect.

Description

LOCAL_RESERVED_DBSPACE_MB lets you control the amount of space Sybase IQ sets aside in your IQ Local Store on a particular query server for certain small but critical data structures used during release savepoint, commit, and checkpoint operations. Set this value to between 200MB and 1GB. The larger your IQ page size and number of concurrent connections, the more reserved space you need.

Sybase IQ reserves the minimum of 200MB or 50% of the size of the last dbspace, which helps DBAs avoid out-of-space conditions by reserving more space automatically.

See also

Reserving space to handle out-of-space conditions in Chapter 5, "Working with Database Objects" in the *Sybase IQ System Administration Guide*.

LOG_CONNECT option

Function	Controls logging of user connections.
Allowed values	ON, OFF
Default	ON
Scope	Can be set only for the PUBLIC group. Takes effect immediately.
Description	When this option is ON, a message appears in the IQ message log (*.iqmsg* file) every time a user connects to or disconnects from the Sybase IQ database.

Note If this option is set OFF (connection logging disabled) when a user connects, and then turned on before the user disconnects, the message log shows that user disconnecting but not connecting.

LOG_CURSOR_OPERATIONS option

Function	Controls logging of cursor operations.
Allowed values	ON, OFF
Default	OFF
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	When this option is ON, a message appears in the IQ message log every time you open or close a cursor. Normally this option should be OFF, which is the default. Turn it ON only if you are having a problem and must provide debugging data to Sybase Technical Support.

LOGIN_MODE option

Function	Controls the use of integrated logins for the database.
Allowed values	Standard, Mixed, or Integrated
Default	Standard
Scope	Can be set only for the PUBLIC group. Takes effect immediately.
Description	This option specifies whether integrated logins are permitted. Values are case insensitive:

- Standard – The default setting, which does not permit integrated logins. An error occurs if an integrated login connection is attempted.
- Mixed – Both integrated logins and standard logins are allowed.
- Integrated – With this setting, all logins to the database must be made using integrated logins.

Warning! Setting the LOGIN_MODE database option to Integrated restricts connections to only those users who have been granted an integrated login mapping. Attempting to connect using a user ID and password generates an error. The only exceptions to this are users with DBA authority (full administrative rights).

See also

For more information on integrated logins, see Chapter 3, "Sybase IQ Connections" in the *Sybase IQ System Administration Guide*.

LOGIN_PROCEDURE option

Function

Specifies a login procedure that sets connection compatibility options at start-up.

Allowed values

String

Default

DBA.sp_iq_process_login

Scope

Can be set for an individual connection or the PUBLIC group. Requires DBA permissions to set the option. Takes effect immediately.

Description

The default login procedure, sp_iq_process_login, executes when a user attempts to connect.

- When Sybase IQ Login Management is enabled, this procedure checks that the user is not locked out, that the maximum number of connections for the user and database is not exceeded, and that the user's password has not expired. It then either allows login to proceed, or sends an error message.
- When Sybase IQ Login Management is disabled, this procedure allows login to proceed.
- If sp_iq_process_login allows login to proceed, it calls the sp_login_environment procedure, which calls to determine the database connection settings.

- In its turn, sp_login_environment checks to see if the connection is being made over TDS. If it is, it calls the sp_tsql_environment procedure, which sets several options to new default values for the current connection.

To use the Login Management facility, LOGIN_PROCEDURE must be set to DBA.sp_iq_process_login.

You can also customize the default database option settings by creating a new procedure and setting LOGIN_PROCEDURE to call that new procedure. *Do not* edit sp_iq_process_login, sp_login_environment or sp_tsql_environment. The customized login procedure must be created in every database you might use.

The Sybase jConnect driver and the iAnywhere ODBC driver reset certain options in accordance with the ODBC specification. They will overwrite settings by the LOGIN_PROCEDURE option for the following:

- Time_format = 'hh:nn:ss'
- Timestamp_format = 'yyyy-mm-dd hh:nn:ss.ssssss'
- Date_format = 'yyyy-mm-dd'
- Date_order = 'ymd'
- Isolation_level = 0

These options will overwrite settings by the LOGIN_PROCEDURE database option. Because these option settings are mandated by the ODBC specification, ODBC applications, including dbisqlapplications, must explicitly set these options if they want different behavior. This could be done using the ODBC connection parameter InitString, for example:

```
iqdsn -wu foo -c
"uid=dba;pwd=sql;eng=foo;InitString='SET OPTION
PUBLIC.DATE_ORDER = ''DMY '''"
```

Example

The following example shows an alternative to sp_iq_process_login. This example disallows a connection by signaling the INVALID_LOGON error.

```
create procedure DBA.login_check()
begin
  declare INVALID_LOGON exception for sqlstate '28000';
  // Allow a maximum of 3 concurrent connections
  if( db_property('ConnCount') > 3 ) then
  signal INVALID_LOGON;
  else
  call sp_login_environment;
  end if;
end
go
```

```
grant execute on DBA.login_check to PUBLIC
go
set option PUBLIC.LOGIN_PROCEDURE='DBA.login_check'
go
```

An alternative means to disallow a connection is by using the RAISERROR statement:

```
CREATE MESSAGE 28000 AS 'User %1! is not allowed to
connect there are already %2! users logged on';
ALTER procedure DBA.login_check()
begin
  declare INVALID_LOGON exception for sqlstate '28000';
  // Allow a maximum of 3 concurrent connections
  if( db_property('ConnCount') > 2 ) then
    RAISERROR 28000, connection_property('Userid'),
      db_property('ConnCount')
  else
    call sp_login_environment;
  end if;
end
```

See also

"Initial option settings" on page 29.

"sp_iq_process_login procedure" on page 819.

Managing IQ user accounts and connections in Chapter 12, "Managing User IDs and Permissions" in the *Sybase IQ System Administration Guide*.

MAIN_CACHE_MEMORY_MB option

Function

Specifies the size of the main shared buffer cache.

Allowed values

1 – 4294967295 (2^{32} -1)

Default

16

Scope

Can be set only for the PUBLIC group. Requires DBA permissions to set the option. Shut down and restart the database server for the change to take effect.

Description

This option sets the size of the main shared memory buffer cache for the database. Sybase recommends that you do not use this option; instead, set the main buffer cache size with the -iqmc server option.

On 64-bit systems, you can allocate as much physical memory as you have to IQ buffer caches; however, for values greater than 4GB, you must use the server options -iqmc and -iqtc to set main and temporary buffer cache sizes. On 32-bit systems, the operating system limits the amount of memory you can allocate. See the *Sybase IQ Installation and Configuration Guide* for your platform for details.

For any active database, the default main buffer cache size of 16MB is too low. For optimal performance, allocate as much memory as possible to the IQ main and temporary buffer caches. For example, if you have 4GB of shared memory on your machine available to Sybase IQ, you can split that amount between the main and temporary shared buffer caches.

When setting the main cache size you must consider many factors, including total physical memory, swap space, memory for the temporary buffer cache, your mix of query and load processing, as well as memory requirements of the operating system and other applications on the machine.

See also

For information about setting buffer cache sizes, see Chapter 5, "Managing System Resources" in the *Sybase IQ Performance and Tuning Guide*.

MAIN_KB_PER_STRIPE option

Function

Defines the number of kilobytes (KB) to write to each dbspace before the disk striping algorithm moves to the next stripe for the IQ Main Store.

Allowed values

Integer greater than zero, in kilobytes

Default

1 (which rounds up to one page)

Scope

Can be set only for the PUBLIC group. Requires DBA permissions to set the option. Takes effect at the next checkpoint.

Description

MAIN_KB_PER_STRIPE lets you control the number of kilobytes written to each dbspace before the IQ disk striping algorithm moves to the next stripe for the IQ Main Store. The corresponding number of blocks is rounded up to a page boundary, so the actual amount written to each stripe might be slightly larger than requested. You can tune this option by measuring the time required to complete I/O intensive updates and adjusting the option value accordingly.

See also

Chapter 5, "Working with Database Objects" in the *Sybase IQ System Administration Guide*.

"Balancing I/O" on page 135 in Chapter 5, "Managing System Resources" in the *Sybase IQ Performance and Tuning Guide*.

MAIN_RESERVED_DBSPACE_MB option

Function	Controls the amount of space Sybase IQ reserves in the IQ Main Store.
Allowed values	Integer greater than zero, in megabytes
Default	200; Sybase IQ actually reserves the minimum of 200MB and 50% of the size of the last dbspace
Scope	Can be set only for the PUBLIC group. Requires DBA permissions to set the option. Shut down and restart the database server for the change to take effect.
Description	MAIN_RESERVED_DBSPACE_MB lets you control the amount of space Sybase IQ sets aside in your IQ Main Store for certain small but critical data structures used during release savepoint, commit, and checkpoint operations. For a production database, set this value to between 200MB and 1GB. The larger your IQ page size and number of concurrent connections, the more reserved space you need. Sybase IQ reserves the minimum of 200MB and 50% of the size of the last dbspace, which helps DBAs avoid out-of-space conditions by reserving more space automatically.
See also	Reserving space to handle out-of-space conditions in Chapter 5, “Working with Database Objects” in the *Sybase IQ System Administration Guide*.

MAX_CARTESIAN_RESULT option

Function	Limits the number of rows resulting from a Cartesian join.
Allowed values	Any integer Can be set temporary (for a connection), for a user, or for the PUBLIC group. Takes effect immediately.
Default	10000000
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	MAX_CARTESIAN_RESULT limits the number of result rows from a query containing a Cartesian join (usually the result of missing one or more join conditions when creating the query). If Sybase IQ cannot find a query plan for the Cartesian join with an estimated result under this limit, it rejects the query and returns an error. Setting MAX_CARTESIAN_RESULT to 0 disables the check for the number of result rows of a Cartesian join.

MAX_CLIENT_NUMERIC_PRECISION option

Function	Controls the maximum precision for numeric data sent to the client.
Allowed values	0 – 126
Default	0
Scope	Can be set by any user, at any level. This option takes effect immediately.

Description	When Sybase IQ performs its calculation, it promotes data types to an appropriate size that ensures accuracy. The promoted data type might be larger in size than Open Client and some ODBC applications can handle correctly.
	
	When MAX_CLIENT_NUMERIC_PRECISION is a nonzero value, Sybase IQ checks that numeric result columns do not exceed this value. If the result column is bigger than MAX_CLIENT_NUMERIC_PRECISION allows, and Sybase IQ is unable to cast it to the specified precision, the query returns the error:
	
	`Data Exception - data type conversion is not possible %1`
	`SQLCODE = -1001006`

See also	"MAX_CLIENT_NUMERIC_SCALE option" on page 111. To control precision for queries on the Catalog Store, see "PRECISION option" on page 135.

MAX_CLIENT_NUMERIC_SCALE option

Function	Controls the maximum scale for numeric data sent to the client.
Allowed values	0 – 126
Default	0
Scope	Can be set by any user, at any level. This option takes effect immediately.
Description	When Sybase IQ performs its calculation, it promotes data types to an appropriate scale and size that ensure accuracy. The promoted data type might be larger than the original defined data size. You can set this option to the scale you want for numeric results. Multiplication, division, addition, subtraction, and aggregate functions can all have results that exceed the maximum precision and scale.

For example, when a DECIMAL(88,2) is multiplied with a DECIMAL(59,2), the result could require a DECIMAL(147,4). With MAX_CLIENT_NUMERIC_PRECISION of 126, only 126 digits are kept in the result. If MAX_CLIENT_NUMERIC_SCALE is 4, the results are returned as a DECIMAL(126,4). If MAX_CLIENT_NUMERIC_SCALE is 2, the result are returned as a DECIMAL(126,2). In both cases, there is a possibility for overflow.

See also

"MAX_CLIENT_NUMERIC_PRECISION option" on page 111.

To control scale for queries on the Catalog Store, see "SCALE option" on page 145.

MAX_CUBE_RESULT option

Function

Sets the maximum number of rows that the IQ optimizer considers for a GROUP BY CUBE operation.

Allowed values

0 – 250000000

Default

10000000

Scope

Can be set by any user, at any level. This option takes effect immediately.

Description

When generating a query plan, the IQ optimizer estimates the total number of groups generated by the GROUP BY CUBE hash operation. The IQ optimizer uses a hash algorithm for the GROUP BY CUBE operation. This option sets an upper boundary for the number of estimated rows the optimizer considers for a hash algorithm that can be run. If the actual number of rows exceeds the MAX_CUBE_RESULT option value, the optimizer stops processing the query and returns the error message "Estimate number: *nnn* exceed the DEFAUTL_MAX_CUBE_RESULT of GROUP BY CUBE or ROLLUP", where *nnn* is the number estimated by the IQ optimizer.

Set MAX_CUBE_RESULT to zero to override the default value. When this option is set to zero, the IQ optimizer does not check the row limit and allows the query to run. Setting MAX_CUBE_RESULT to zero is not recommended, as the query might not succeed.

MAX_CURSOR_COUNT option

Function	Specifies a resource governor to limit the maximum number of cursors that a connection can use at once.
Allowed values	Integer
Default	50
Scope	Can be set for an individual connection or the PUBLIC group. Takes effect immediately. Requires DBA permissions to set this option for *any* connection.
Description	The specified resource governor allows a DBA to limit the number of cursors per connection that a user can have. If an operation exceeds the limit for a connection, an error is generated indicating that the limit has been exceeded. If a connection executes a stored procedure, that procedure is executed under the permissions of the procedure owner. However, the resources used by the procedure are assigned to the current connection. You can remove resource limits by setting MAX_CURSOR_COUNT to 0 (zero).

MAX_HASH_ROWS option

Function	Sets the maximum number of rows that the IQ optimizer considers for a hash algorithm.
Allowed values	Integer up to 250000000
Default	2500000
Scope	Can be set temporary, per user, or for the PUBLIC group. DBA permissions are not required to set the option. This option takes effect immediately.
Description	When generating a query plan, the IQ optimizer might have several algorithms (hash, sort, indexed) to choose from when processing a particular part of a query. These choices often depend on estimates of the number of rows to process or generate from that part of the query. This option sets an upper boundary for how many estimated rows are considered for a hash algorithm. For example, if there is a join between two tables, and the estimated number of rows entering the join from both tables exceeds the value of MAX_HASH_ROWS, the optimizer does not consider a hash join. On systems with more than 50 MB per user of temporary buffer cache space, you might want to consider a higher value for this option.

MAX_IQ_THREADS_PER_CONNECTION option

Function	Controls the number of threads for each connection.
Allowed values	3 – 1000
Default	72
Scope	Can be temporary or permanent. Requires DBA permissions to set. Can be set for the PUBLIC group only. Takes effect immediately.
Description	Allows you to constrain the number of threads (and thereby the amount of system resources) the commands executed on a connection use. For most applications, use the default.

MAX_IQ_THREADS_PER_TEAM option

Function	Controls the number of threads allocated to perform a single operation (such as a LIKE predicate on a column) executing within a connection.
Allowed values	1 – 1000
Default	48
Scope	Can be temporary or permanent. Requires DBA permissions to set. Can be set for the PUBLIC group only. Takes effect immediately.
Description	Allows you to constrain the number of threads (and thereby the amount of system resources) allocated to a single operation. The total for all simultaneously executing teams for this connection is limited by the related option, MAX_IQ_THREADS_PER_CONNECTION. For most applications, use the default.

MAX_JOIN_ENUMERATION option

Function	Controls the maximum number of tables to be optimized for join order after optimizer simplifications have been applied.
Allowed values	1 – 64
Default	15
Scope	Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

The query optimizer simplifies its optimization of join order by separate handling of both lookup tables (that is, nonselective dimension tables) and tables that are effective Cartesian products. After simplification, it proceeds with optimizing the remaining tables for join order, up to the limit set by MAX_JOIN_ENUMERATION. If this limit is exceeded, the query is rejected with an error. The user can then either simplify the query or try increasing the limit.

Normally, you should not need to change this value. If you do, Sybase recommends setting MAX_JOIN_ENUMERATION as a temporary or user option.

MAX_QUERY_PARALLELISM option

Function

Sets upper bound for parallel execution of GROUP BY operations and for arms of a UNION.

Allowed values

Integer less than or equal to number of CPUs.

Default

24

Scope

Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

Sets an upper bound for parallelism the query optimizer can choose for GROUP BY operations or arms of a UNION, regardless of how many CPUs are available. This option is effective only on GROUP BY operations when PARALLEL_GBH_UNITS is *not* set. The PARALLEL_GBH_UNITS option sets a specific number for the degree of parallelism, whereas the MAX_QUERY_PARALLELISM option value is an upper limit and allows the optimizer more flexibility.

Normally, you should not set this option. However, if you have more than 16 CPUs and you see excessive CPU time spent on system usage, try setting MAX_QUERY_PARALLELISM to a value less than 16. Experiment with this value to determine the right setting for your platform, number of CPUs, and queries. There is some overhead involved when you distribute execution across multiple CPUs. In some configurations, this overhead may actually decrease performance if parallelism is allowed across all available CPUs, while in others, using all available CPUs may be beneficial.

See also

“PARALLEL_GBH_UNITS option” on page 133.

MAX_QUERY_TIME option

Function	Sets a time limit so that the optimizer can disallow very long queries.
Allowed values	0 to 2^{32} - 1 minutes
Default	0 (disabled)
Scope	Can be set at the session (temporary), user, or PUBLIC level.
Description	If the query runs longer than the MAX_QUERY_TIME setting, Sybase IQ stops the query and sends a message to the user and the IQ message file. For example: `The operation has been cancelled -- Max_Query_Time exceeded.` MAX_QUERY_TIME applies only to queries and not to any SQL statement that is modifying the contents of the database.

MAX_STATEMENT_COUNT option

Function	Specifies a resource governor to limit the maximum number of prepared statements that a connection can use at once.
Allowed values	Integer
Default	100
Scope	Can be set for an individual connection or the PUBLIC group. Takes effect immediately. Requires DBA permissions to set this option for *any* connection.
Description	The specified resource governor allows a DBA to limit the number of prepared statements per connection that a user can have. If an operation exceeds the limit for a connection, an error is generated indicating that the limit has been exceeded. If a connection executes a stored procedure, that procedure is executed under the permissions of the procedure owner. However, the resources used by the procedure are assigned to the current connection. You can remove resource limits by setting MAX_STATEMENT_COUNT to 0 (zero).

MAX_WARNINGS option

Function	Controls the maximum number of warnings allowed.
Allowed values	Any integer
Default	$2^{64} - 1$
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	This option can limit the number of warnings about rejected values, row mismatches, and so on during DDL commands. The default does not restrict the number you can receive.

MINIMIZE_STORAGE option

Function	Minimize use of disk space for newly created columns.
Allowed Values	ON, OFF
Default	OFF
Scope	Can be set for the PUBLIC group or for temporary use. DBA authority is required to set the option. This option takes effect immediately.
Description	When MINIMIZE_STORAGE is ON, IQ optimizes storage for new columns by using as little as one byte of disk space per row wherever appropriate. By default, this option is OFF for the PUBLIC group, and one-byte storage is used for all newly created columns; when it is OFF for the PUBLIC group but ON as a temporary user option, one-byte storage is used for new columns created by that user ID. MINIMIZE_STORAGE=ON is equivalent to placing an IQ UNIQUE 255 clause on every new column, with the exception of certain data types that are by nature too wide for one-byte storage. When MINIMIZE_STORAGE is ON, there is no need to specify IQ UNIQUE except for columns with more than 65536 unique values. Tables with few columns benefit when MINIMIZE_STORAGE is ON. Tables with many columns generally benefit from turning this option OFF. The definition of “few” depends on the processor; for larger processors, the number can be greater than for smaller ones. Specifying IQ UNIQUE explicitly in CREATE TABLE or ALTER TABLE ADD COLUMN overrides the MINIMIZE_STORAGE option for that column.

See also — Chapter 5, "Working with Database Objects" in *Sybase IQ System Administration Guide*.

MIN_NLPDJ_FILTERED_PPM option

Function — Constrains the join algorithm choices available to the optimizer under certain circumstances.

Allowed values — UNSIGNED INT1 – 1,000,000

Default — 2500

Scope — Can be set temporary, for a user, or for the PUBLIC group. Takes effect immediately.

Description — Specifies the minimum percentage of rows that must remain after all simple local predicates (expressed in parts-per-million) before the optimizer will consider using the nested-loop push-down join (NLPDJ) algorithm. The default is equivalent to a selectivity of 0.0025, or one quarter of one percent of the table.

This option only affects the optimizer when the target table is very large. Under normal circumstances, you should not need to change this option.

MIN_NLPDJ_TABLE_SIZE option

Function — Specifies the minimum number of rows that must be present in a table before the join optimizer considers using the nested-loop push-down join (NLPD) algorithm.

Allowed values — 1 – 4294967295

Default — 10000

Scope — Can be set temporary, for a user or for the PUBLIC group. Takes effect immediately.

Description — This option allows you to control the minimum number of rows in a table before the join optimizer considers using the nested-loop push-down join algorithm. Under most circumstances, you do not need to change the value of this option.

MIN_PASSWORD_LENGTH option

Function	Sets the minimum length for new passwords in the database.
Allowed values	Integer greater than or equal to zero The value is in bytes. For single-byte character sets, this is the same as the number of characters.
Default	0 characters
Scope	Can be set for the PUBLIC group. Takes effect immediately. Requires DBA permissions to set this option.
Description	This option allows the DBA to impose a minimum length on all new passwords for greater security. Existing passwords are not affected.
Example	• Sets the minimum length for new passwords to 6 bytes: `SET OPTION PUBLIC.MIN_PASSWORD_LENGTH = 6`

MIN_SMPDJ_OR_HPDJ_FILTERED_PPM option

Function	Constrains the join algorithm choices available to the optimizer under certain circumstances.
Allowed values	UNSIGNED INT1 – 1,000,000
Default	2500
Scope	Can be set temporary, for a user, or for the PUBLIC group. Takes effect immediately.
Description	Specifies the minimum percentage of rows that must remain after all simple local predicates (expressed in parts-per-million) before the optimizer considers using either the hash push-down join (HPDJ) or sort-merge push-down join (SMPDJ) algorithms. The default is equivalent to a selectivity of 0.0025, or one quarter of one percent of the table. This option affects the optimizer only when the target table is very large. Under normal circumstances, you should not need to change this option.

MIN_SMPDJ_OR_HPDJ_FILTERED_SIZE option

Function	Constrains the join algorithm choices available to the optimizer under certain circumstances.
Allowed values	1 – 4294967295
Default	25000
Scope	Can be set temporary, for a user or for the PUBLIC group. Takes effect immediately.
Description	MIN_SMPDJ_OR_HPDJ_FILTERED_SIZE allows you to control the minimum number of rows a table (or a UNION ALL view of tables) must have left after the filtering effects of all local predicates have been considered before the join optimizer considers using either the hash push-down join (HPDJ) or the sort-merge push-down join (SMPDJ) algorithms for situations where there are no joins or only lookup joins between this join and the table (or the UNION ALL view). Under most circumstances, you do not need to change the value of this option.

MIN_SMPDJ_OR_HPDJ_INDIRECT_SIZE option

Function	Constrains the join algorithm choices available to the optimizer under certain circumstances.
Allowed values	1 – 4294967295
Default	500000
Scope	Can be set temporary, for a user or for the PUBLIC group. Takes effect immediately.
Description	MIN_SMPDJ_OR_HPDJ_INDIRECT_SIZE allows you to control the minimum number of rows a table (or a UNION ALL view of tables) must have left after the filtering effects of all local predicates have been considered before the join optimizer considers using either the hash push-down join (HPDJ) or the sort-merge push-down join (SMPDJ) algorithms for situations where there are nonlookup (many-to-many or many-to-1) joins between this join and the table (or the UNION ALL view). Under most circumstances, you do not need to change the value of this option.

MIN_SMPDJ_OR_HPDJ_TABLE_SIZE option

Function	Constrains the join algorithm choices available to the optimizer under certain circumstances.
Allowed values	1 – 4294967295
Default	100000
Scope	Can be set temporary, for a user or for the PUBLIC group. Takes effect immediately.
Description	MIN_SMPDJ_OR_HPDJ_TABLE_SIZE allows you to control the minimum number of rows that must be present in a table (or a UNION ALL view of tables) before the join optimizer considers using either the hash push-down join (HPDJ) or the sort-merge push-down join (SMPDJ) algorithms. Under most circumstances, you do not need to change the value of this option.

MONITOR_OUTPUT_DIRECTORY option

Function	The MONITOR_OUTPUT_DIRECTORY option controls placement of output files for the IQ buffer cache monitor. All monitor output files are used for the duration of the monitor runs, which cannot exceed the lifetime of the connection. The output file still exists after the monitor run stops. A connection can run up to two performance monitors simultaneously, one for main cache and one for temp cache. A connection can run a monitor any number of times, successively. MONITOR_OUTPUT_DIRECTORY controls the directory in which the monitor output files are created, regardless of what is being monitored or what monitor mode is used.
Allowed values	String.
Default	Same directory as the database.
Scope	Can be set for the PUBLIC group. Takes effect immediately. Requires DBA permissions to set this option.
Description	The IQ monitor sends output to the directory specified by this option. The dummy table used to start the monitor can be either a temporary or a permanent table. The directory can be on any physical machine. The DBA can use the PUBLIC setting to place all monitor output in the same directory, or set different directories for individual users.

Example

This example shows how you could declare a temporary table for monitor output, set its location, and then have the monitor start sending files to that location for the main and temp buffer caches.

Note In this example, the output directory string is set to both "/tmp" and "tmp/". The trailing slash ("/") is correct and is supported by the interface. The example illustrates that the buffer cache monitor does not require a permanent table; a temporary table can be used.

```
declare local temporary table dummy_monitor
(dummy_column integer)

set option Monitor_Output_Directory = "/tmp"
iq utilities main into dummy_monitor start monitor '-
debug -interval 2'

set option Monitor_Output_Directory = "tmp/"
iq utilities private into dummy_monitor start monitor
'-debug -interval 2'
```

MPX_GLOBAL_TABLE_PRIV option

Function

Lets a query server grant and revoke permissions on objects created by the write server.

Allowed values

ON, OFF

Scope

DBA permissions are required to set this option. Can be set only for the PUBLIC group. Takes effect immediately.

Default

OFF

Description

To enable this option, set it ON. Setting MPX_GLOBAL_TABLE_PRIV ON allows grant and revoke of table and execute permissions of write server objects on a query server.

See also

"MPX_LOCAL_SPEC_PRIV option" on page 123.

MPX_LOCAL_SPEC_PRIV option

Function
Lets a query server create and drop users, groups, and group memberships.

Allowed values
0 – 63

A bitmask indicating the corresponding special privileges to be granted and revoked on a query server:

- 0x01 = DBA
- 0x02 = create user through GRANT CONNECT and drop user through REVOKE CONNECT
- 0x04 = RESOURCE
- 0x10 = GROUP
- 0x20 = MEMBERSHIP

To combine two or more privileges, add the bitmasks in hexadecimal (base 16), then convert to decimal to determine the value for the option. For example, to combine RESOURCE and GROUP privileges on a database, use the formula `4 + 16 (10 in base 16) = 20` and set MPX_LOCAL_SPEC_PRIV to 20.

To allow all privileges, you must set all bits. To do this, set MPX_LOCAL_SPEC_PRIV to 63.

Scope
Can be set only for the PUBLIC group. Takes effect immediately.

Default
0

Description
To enable MPX_LOCAL_SPEC_PRIV, set it to the appropriate value between 1 and 63. See “Allowed values” on page 122. DBA permissions are required to set this option. This option takes effect immediately.

See also
“MPX_GLOBAL_TABLE_PRIV option” on page 122.

NEAREST_CENTURY option [TSQL]

Function
Controls the interpretation of 2-digit years, in string to date conversions.

Allowed values
0 – 100

Default
50

Description
NEAREST_CENTURY controls the handling of 2-digit years, when converting from strings to dates or timestamps.

The NEAREST_CENTURY setting is a numeric value that acts as a rollover point. Two-digit years less than the value are converted to 20yy, whereas years greater than or equal to the value are converted to 19yy.

Adaptive Server Enterprise and Sybase IQ behavior is to use the nearest century, so that if the year value *yy* is less than 50, then the year is set to 20yy.

NOEXEC option

Function	Generates the optimizer query plans instead of executing the plan.
Allowed values	ON, OFF
Default	OFF
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	When determining how to process a query, the IQ optimizer generates a query plan to map how it plans to have the query engine process the query. If this option is set ON, the optimizer sends the plan for the query to the IQ message file rather than submitting it to the query engine. This option affects only queries or commands that include a query. Note that when the EARLY_PREDICATE_EXECUTION option is ON, IQ executes the local predicates for all queries before generating a query plan, even when the NOEXEC option is ON. The generated query plan is the same as the runtime plan.
See also	“EARLY_PREDICATE_EXECUTION option” on page 74.

NON_ANSI_NULL_VARCHAR option

Function	Controls whether zero-length varchars are treated as NULLs for insert/load/update purposes.
Allowed values	ON, OFF
Default	OFF
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

NON_ANSI_NULL_VARCHAR lets you revert to non-ANSI (Version 12.03.1) behavior for treating zero-length VARCHAR data during load or update operations. When this option is set to OFF, zero-length varchars are stored as zero-length during load, insert, or update. When this option is set to ON, zero-length VARCHAR data is stored as NULLs on load, insert, or update.

NON_KEYWORDS option [TSQL]

Function

Turns off individual keywords, allowing their use as identifiers.

Allowed values

String

Default

" (the empty string)

Description

NON_KEYWORDS turns off individual keywords. If you have an identifier in your database that is now a keyword, you can either add double quotes around the identifier in all applications or scripts, or you can turn off the keyword using the NON_KEYWORDS option.

The following statement prevents TRUNCATE and SYNCHRONIZE from being recognized as keywords:

```
SET OPTION NON_KEYWORDS = 'TRUNCATE, SYNCHRONIZE'
```

Each new setting of this option replaces the previous setting. This statement clears all previous settings:

```
SET OPTION NON_KEYWORDS =
```

A side effect of the options is that SQL statements using a turned-off keyword cannot be used; they produce a syntax error.

NOTIFY_MODULUS option

Function

Controls the default frequency of notify messages issued by certain commands.

Allowed values

Any integer

Default

100000

Scope

DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description — This option sets the default number of notify messages Sybase IQ issues for certain commands that produce them. The NOTIFY clause for some of the commands (such as CREATE INDEX, LOAD TABLE, and DELETE) override this value. Other commands that do not support the NOTIFY clause (such as SYNCHRONIZE JOIN INDEX) always use this value. The default does not restrict the number of messages you can receive.

NULLS option [DBISQL]

Function — Specifies how NULL values in the database are displayed.

Allowed values — ON, OFF

Default — ON (NULL)

Description — Set this according to your preference.

ODBC_DISTINGUISH_CHAR_AND_VARCHAR option

Function — Controls how the Sybase IQ and Adaptive Server Anywhere ODBC driver describes CHAR columns.

Allowed values — ON, OFF

Default — OFF

Description — When a connection is opened, the Sybase IQ and Adaptive Server Anywhere ODBC driver uses the setting of this option to determine how CHAR columns are described. If ODBC_DISTINGUISH_CHAR_AND_VARCHAR is set to OFF (the default), then CHAR columns are described as SQL_VARCHAR. If this option is set to ON, then CHAR columns are described as SQL_CHAR. VARCHAR columns are always described as SQL_VARCHAR.

See also — Chapter 4, “SQL Data Types” in *Sybase IQ Reference Manual*.

ON_CHARSET_CONVERSION_FAILURE option

Function — Controls what happens if an error is encountered during character conversion.

Allowed values — String. See Description for allowed values.

Default — IGNORE

Description

Controls what happens if an error is encountered during character conversion, as follows:

- **IGNORE** Errors and warnings do not appear.
- **WARNING** Reports substitutions and illegal characters as warnings. Illegal characters are not translated.
- **ERROR** Reports substitutions and illegal characters as errors.

Single-byte to single-byte converters are not able to report substitutions and illegal characters, and must be set to IGNORE.

ON_ERROR option [DBISQL]

Function

Controls what happens if an error is encountered while executing statements in Interactive SQL.

Allowed values

String. See Description for allowed values.

Default

PROMPT

Description

Controls what happens if an error is encountered while executing statements as follows:

- STOP – DBISQL stops executing statements from the file and returns to the statement window for input.
- PROMPT – DBISQL prompts the user to see if he or she wants to continue.
- CONTINUE – The error displays and DBISQL continues executing statements.
- EXIT – DBISQL terminates.
- NOTIFY_CONTINUE – The error is reported, and the user is prompted to press ENTER or click OK to continue.
- NOTIFY_STOP – The error is reported, and the user is prompted to press ENTER or click OK to stop executing statements.
- NOTIFY_EXIT – The error is reported, and the user is prompted to press ENTER or click OK to terminate Interactive SQL.

When you are executing a *.SQL* file, the values STOP and EXIT are equivalent.

ON_TSQL_ERROR option [TSQL]

Function — Controls error-handling in stored procedures.

Allowed values — String. See Description for allowed values.

Default — CONDITIONAL

Description — This option controls error handling in stored procedures.

- STOP– Stops execution immediately upon finding an error.
- CONDITIONAL – If the procedure uses ON EXCEPTION RESUME, and the statement following the error handles the error, continue, otherwise exit.
- CONDITION – Continue execution, regardless of the following statement. If there are multiple errors, the first error encountered in the stored procedure is returned. This option most closely mirrors Adaptive Server Enterprise behavior.

Both CONDITIONAL and CONTINUE settings for ON_TSQL_ERROR are used for Adaptive Server Enterprise compatibility, with CONTINUE most closely simulating Adaptive Server Enterprise behavior. The CONDITIONAL setting is recommended, particularly when developing new Transact-SQL stored procedures, as it allows errors to be reported earlier.

When this option is set to STOP or CONTINUE, it supersedes the setting of the CONTINUE_AFTER_RAISERROR option. However, when this option is set to CONDITIONAL (the default), behavior following a RAISERROR statement is determined by the setting of the CONTINUE_AFTER_RAISERROR option.

See also — CREATE PROCEDURE statement on page 485.

CREATE PROCEDURE statement [T-SQL] on page 491.

“CONTINUE_AFTER_RAISERROR option [TSQL]” on page 53.

“Transact-SQL procedure language overview” on page 936.

Appendix A, “Compatibility with Other Sybase Databases.’

OS_FILE_CACHE_BUFFERING option

Function — Controls use of file system buffering.

Allowed values — ON, OFF

Default — OFF; default affects newly created databases only.

Scope
Can be set for the PUBLIC group only. You must shut down the database and restart it for the change to take effect. Requires DBA permissions to set this option.

Description
This performance option is available on Solaris UFS file systems and Windows file systems only. It does not affect databases on raw disk.

Setting OS_FILE_CACHE_BUFFERING OFF prevents file system buffering for IQ Store files. Turning off file system buffering saves a data copy from the file system buffer cache to the main IQ buffer cache. Usually this reduces paging caused by competition for memory between the IQ buffer manager and the operating system's file system buffer. When it reduces paging, this option improves performance; however, if the IQ page size for the database is less than the file system's block size (typically only in the case in testing situations), performance *decreases*, especially during multiuser operation.

Experiment with this option to determine the best setting for different conditions. You must restart the database for the new setting to take effect.

See also
Chapter 5, "Managing System Resources" in the *Sybase IQ Performance and Tuning Guide*.

OUT_OF_DISK_MESSAGE_REPEAT option

Function
Controls the interval time between out of disk space messages.

Allowed values
Any integer

Default
120

Scope
Can be set only for the PUBLIC group. DBA authority is required to set the option. You must shut down and restart the database server for the change to take effect.

Description
OUT_OF_DISK_MESSAGE_REPEAT resets the default amount of time at which Sybase IQ should repeat the cycle of checking and then issuing an out-of-space message. To determine this length of time, multiply this option value with the value for OUT_OF_DISK_WAIT_TIME. For example, using the defaults of 120 for OUT_OF_DISK_MESSAGE_REPEAT and 30 seconds for OUT_OF_DISK_WAIT_TIME, Sybase IQ would issue an out-of-space message every hour.

OUT_OF_DISK_WAIT_TIME option

Function	Controls the default interval time to wait before checking again when out of disk space.
Allowed values	Any integer
Default	30 seconds
Scope	Can be set only for the PUBLIC group. DBA authority is required to set the option. You must shut down and restart the database server for the change to take effect.
Description	OUT_OF_DISK_WAIT_TIME resets the default number of seconds Sybase IQ should wait in a sleep state when it is out of disk space. At the end of this time, Sybase IQ checks to see if any space has been added. If none has been added, it returns to a sleep state and waits before checking again. It repeats this process until disk space is added.

OUTPUT_FORMAT option [ISQL]

Function	Sets the output format for the data retrieved by the SELECT statement and redirected into a file, or output using the OUTPUT statement.
Allowed values	String. See Description for allowed values.
Default	ASCII
Description	The valid output formats are:

- **ASCII** The output is an ASCII format file with one row per line in the file. All values are separated by commas, and strings are enclosed in apostrophes (single quotes). The delimiter and quote strings can be changed using the DELIMITED BY and QUOTE clauses. If ALL is specified in the QUOTE clause, then all values (not just strings) are quoted.

  Three other special sequences are also used. The two characters \n represent a newline character; \\ represents a single backslash character, and the sequence \xDD represents the character with hexadecimal code DD.

- **DBASEII** The output is a dBASE II format file with the column definitions at the top of the file. Note that a maximum of 32 columns can be output. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.

- **DBASEIII** The output is a dBASE III format file with the column definitions at the top of the file. Note that a maximum of 128 columns can be output. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.
- **EXCEL** The output is an Excel 2.1 worksheet. The first row of the worksheet contains column labels (or names if there are no labels defined). Subsequent worksheet rows contain the actual table data.
- **FIXED** The output is fixed format, with each column having a fixed width. The width for each column can be specified using the COLUMN WIDTH clause. If this clause is omitted, the width for each column is computed from the data type for the column, and is large enough to hold any value of that data type. No column headings are output in this format.
- **FOXPRO** The output is a FoxPro format file (the FoxPro memo field is different than the dBASE memo field) with the column definitions at the top of the file. Note that a maximum of 128 columns can be output. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.
- **HTML** The output is in the HyperText Markup Language format.
- **LOTUS** The output is a Lotus WKS format worksheet. Column names are the first row in the worksheet. Note that there are certain restrictions on the maximum size of Lotus WKS format worksheets that other software (such as Lotus 1-2-3) can load. There is no limit to the size of file Interactive SQL can produce.
- **SQL** The output is an Interactive SQL INPUT statement required to recreate the information in the table.
- **XML** The output is an XML file encoded in UTF-8 and containing an embedded DTD. Binary values are encoded in CDATA blocks with the binary data rendered as 2-hex-digit strings.

See also OUTPUT statement [DBISQL] on page 605.

OUTPUT_LENGTH option [ISQL]

Function Controls the length used when Interactive SQL exports information to an external file.

Allowed values *Integer*

Default *0* (no truncation)

Description	This option controls the length used when Interactive SQL exports information to an external file (using output redirection with the OUTPUT statement). This option affects only ASCII, HTML, and SQL output formats.
See also	OUTPUT statement [DBISQL] on page 605.

OUTPUT_NULLS option [ISQL]

Function	Controls the way NULL values appear in result sets.
Allowed values	String
Default	'NULL'
Description	This option controls the way NULL values appear in result sets. Every time a NULL value is found in the result set, the string from this option is returned instead. This setting applies to data displayed in Interactive SQL on the Results tab in the Results pane as well as to data in output files generated by the OUTPUT statement. This option affects only ASCII, HTML, and SQL output formats.
See also	OUTPUT statement [DBISQL] on page 605.

PARALLEL_GBH_ENABLED option

Function	Allows GROUP BY operations on a single table to be executed in parallel using all available CPUs, if determined appropriate by the optimizer.
Allowed values	ON, OFF
Default	ON
Scope	Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	The PARALLEL_GBH_ENABLED option causes GROUP BY operations on a single table to be done in parallel using all available CPUs. This option provides significant performance gains for some queries in certain situations. PARALLEL_GBH_ENABLED is ON by default. You can disable this feature by setting PARALLEL_GBH_ENABLED to OFF or constrain the effect by changing the value of the PARALLEL_GBH_UNITS option.
See also	“PARALLEL_GBH_UNITS option” on page 133.

PARALLEL_GBH_MIN_ROWS_PER_UNIT option

Function	Can limit the degree of parallelism chosen by the optimizer for GROUP BY operations.
Allowed values	0 – 4294967295
Default	3000000
Scope	Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	When the PARALLEL_GBH_ENABLED option is ON, the value of PARALLEL_GBH_MIN_ROWS_PER_UNIT can indirectly limit the degree of parallelism chosen within the optimizer for GROUP BY operations by requiring that each unit of work to be done in parallel must have at least this many rows. The default of 3 million rows means that a table must have at least 6 million rows before the optimizer chooses to execute GROUP BY in parallel over that table. The default of 3 million is appropriate for large databases on systems with numerous CPUs. For smaller systems or for servers where GROUP BY operations frequently involve more complex aggregates and grouping expressions, performance of some queries can be improved by setting this option to a lower value, such as 500,000.
See also	"PARALLEL_GBH_ENABLED option" on page 132. "PARALLEL_GBH_UNITS option" on page 133.

PARALLEL_GBH_UNITS option

Function	Overrules the choice of the optimizer on the degree of parallelism of GROUP BY operations.
Allowed values	0 – 100
Default	0
Scope	Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	When the PARALLEL_GBH_ENABLED option is ON, you can constrain the effect of this feature by changing the value of the PARALLEL_GBH_UNITS option. The PARALLEL_GBH_UNITS option overrules the optimizer's choice on the degree of parallelism. Normally, you should not set this option.

The argument to PARALLEL_GBH_UNITS is a value that represents a specific number of parts to break the GROUP BY into in order to execute in parallel. For example, to run in eight parts, use

```
SET TEMPORARY OPTION PARALLEL_GBH_UNITS = 8
```

If PARALLEL_GBH_UNITS is 0, the optimizer chooses the degree of parallelism based on the number of CPUs, the number of users on the server, and the amount of data to be grouped.

See also

"PARALLEL_GBH_ENABLED option" on page 132.

"MAX_QUERY_PARALLELISM option" on page 115.

PERCENT_AS_COMMENT option [TSQL]

Function

Controls the interpretation of the percent (%) character.

Allowed values

ON, OFF

Default

ON

Description

By default, Sybase IQ treats the percent character as a comment marker. However, Sybase recommends that you do not use it as such; use one of the alternative comment markers such as //, /* */, or -- (double dash) instead. The double-dash style is the SQL92 comment delimiter.

Adaptive Server Enterprise treats the % as a modulo operator; it does not support the Sybase IQ mod function. You can set this option to OFF for compatibility with both environments.

Adaptive Server Anywhere treats the percent character exactly as Sybase IQ, that is, as comment by default, but as a modulo operator if you set the PERCENT_AS_COMMENT option to OFF.

Procedures and views created with %-style comments are converted to double-dash comments when they are stored in the catalog. The Sybase Central code editor does not highlight %-style comments. To have your comments highlighted, use one of the other comment delimiters.

Note Existing procedures that contain %-style comments must be re-created before you change the option setting; otherwise, the procedures fail to load.

PRECISION option

Function	Specifies the maximum number of digits in the result of any decimal arithmetic, for queries on the Catalog Store only.
Allowed values	126
Default	126
Description	Precision is the total number of digits to the left and right of the decimal point. The default PRECISION value is fixed at 126. SCALE specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum specified by PRECISION, for queries on the Catalog Store.
See also	"SCALE option" on page 145. For queries on the IQ Store, see "MAX_CLIENT_NUMERIC_PRECISION option" on page 111.

PREFETCH option

Function	Allows you to turn fetching on or off or to use the ALWAYS value to prefetch the cursor results even for SENSITIVE cursor types and for cursors that involve a proxy table.
Allowed values	ON, OFF, ALWAYS
Default	ON
Scope	Can be set for an individual connection or the PUBLIC group. Takes effect immediately.
Description	For the Catalog Store only, PREFETCH controls whether rows are fetched to the client side before being made available to the client application. Fetching a number of rows at a time, even when the client application requests rows one at a time (for example, when looping over the rows of a cursor) minimizes response time and improves overall throughput by limiting the number of requests to the database. The setting of PREFETCH is ignored by Open Client and JDBC connections, and for the IQ Store.

PREFETCH_BUFFER_LIMIT option

Function	Specifies the amount of memory used for prefetching.
Allowed values	Integer
Default	0
Scope	Can be set only for the PUBLIC group. DBA authority is required to set the option. Shut down and restart the database server to have the change take effect.
Description	PREFETCH_BUFFER_LIMIT defines the number of cache pages available to Sybase IQ for use in prefetching (the read-ahead of database pages). Do not set this option unless advised to do so by Sybase Technical Support.

PREFETCH_BUFFER_PERCENT option

Function	Specifies the percent of memory used for prefetching.
Allowed values	0 – 100
Default	40
Scope	Can be set only for the PUBLIC group. DBA authority is required to set the option. Shut down and restart the database server to have the change take effect.
Description	PREFETCH_BUFFER_PERCENT is an alternative to PREFETCH_BUFFER_LIMIT, as it specifies the percentage of cache available for use in prefetching. Do not set this option unless advised to do so by Sybase Technical Support.

PREFETCH_GARRAY_PERCENT option

Function	Specifies the percent of prefetch resources designated for inserts to HG indexes.
Allowed values	0 – 100
Default	60
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

As with PREFETCH_SORT_PERCENT, this option designates a percentage of prefetch resources for use when inserting into an HG index.

Do not set this option unless advised to do so by Sybase Technical Support.

PREFETCH_SORT_PERCENT option

Function

Specifies the percent of prefetch resources designated for sorting objects.

Allowed values

0 – 100

Default

50

Scope

DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

PREFETCH_SORT_PERCENT designates a percentage of prefetch resources for use by a single sort object. Increasing this value can improve the single-user performance of inserts and deletes, but may have detrimental effects on multiuser operations.

Do not set this option unless advised to do so by Sybase Technical Support.

PRESERVE_SOURCE_FORMAT option [database]

Function

Controls whether the original source definition of procedures, views, and event handlers is saved in system files. If saved, it is saved in the column source in SYSTABLE, SYSPROCEDURE, and SYSEVENT.

Allowed values

ON, OFF

Default

ON

Description

When PRESERVE_SOURCE_FORMAT is ON, the server saves the formatted source from CREATE and ALTER statements on procedures, views, and events, and puts it in the appropriate system table's source column.

Unformatted source text is stored in the same system tables, in the columns proc_defn, and view_defn. However, these definitions are not easy to read in Sybase Central. The formatted source column allows you to view the definitions with the spacing, comments, and case that you want.

This option can be turned off to reduce space used to save object definitions in the database. The option can be set only for the user PUBLIC.

QUERY_DETAIL option

Function	Specifies whether or not to include additional query information.
Allowed values	ON, OFF
Default	OFF
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	When QUERY_DETAIL and QUERY_PLAN (or QUERY_PLAN_AS_HTML) are both turned on, Sybase IQ displays additional information about the query when producing its query plan. When QUERY_PLAN and QUERY_PLAN_AS_HTML are OFF, this option is ignored. When QUERY_PLAN is ON (the default), especially if QUERY_DETAIL is also ON, you might want to enable message log wrapping to avoid filling up your message log file. See "IQMSG_LENGTH_MB option" on page 92 for details.
See also	"QUERY_PLAN option" on page 139. "QUERY_PLAN_AS_HTML option" on page 140.

QUERY_NAME option

Function	Gives a name to an executed query.
Allowed values	Quote-delimited string of up to 80 characters.
Default	'' (the empty string)
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description	You can assign the QUERY_NAME option any quote-delimited string value, up to 80 characters; for example:
	
	`set temporary option Query_Name = 'my third query'`
	
	When this option is set, query plans that are sent to the *.iqmsg* file or *.html* file include a line near the top of the plan that looks like:
	
	`Query_Name:  'my third query'`
	
	If you set the option to a different value before each query in a script, it is much easier to identify the correct query plan for a particular query. Doing this also prevents previous query plans from being overwritten. This option has no other effect on the query.

QUERY_PLAN option

Function	Specifies whether or not additional query messages are produced.
Allowed values	ON, OFF
Default	ON
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	When this option is turned ON, Sybase IQ produces messages about queries. These include messages about using join indexes, the join order, join algorithms for the queries, and columns being extracted using the data extraction options. When this option is turned OFF, those messages are suppressed. The information is sent to the <*dbname*>.*iqmsg* file.
See also	“QUERY_DETAIL option” on page 138. “QUERY_PLAN_AS_HTML option” on page 140. “QUERY_PLAN_AFTER_RUN option” on page 139.

QUERY_PLAN_AFTER_RUN option

Function	Prints the entire query plan after query execution is complete.
Allowed values	ON, OFF
Default	OFF
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	When QUERY_PLAN_AFTER_RUN is turned ON, the query plan is printed after the query has finished running. This allows the query plan to include additional information, such as the actual number of rows passed on from each node of the query. For this option to work, the QUERY_PLAN option must be set to ON (the default). You can use this option in conjunction with QUERY_DETAIL to generate additional information in the query plan report.

QUERY_PLAN_AS_HTML option

Function	Generates graphical query plans in HTML format for viewing in a Web browser.
Allowed values	ON, OFF
Default	OFF
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

QUERY_PLAN_AS_HTML causes graphical query plans to be generated in HTML format.

When you set this option, also set the QUERY_NAME option for each query, so you know which query is associated with the query plan.

Sybase IQ writes the plans in the same directory as the *.iqmsg* file, in a file named:

user-name_query-name_YYYYMMDD_HHMMSS.html

For example, if the user DBA sets the temporary option QUERY_NAME to `'Query_1123'`, a file created on April 18, 2002 at exactly 8:30 a.m. is called *DBA_Query_1123_20020418_083000.html*. The date and time are appended to the file name automatically to ensure that existing files are not overwritten.

Note If you use this feature, monitor your disk space usage so you leave enough room for your *.iqmsg* and log files to grow. Enabling IQ message log wrapping helps control the size of this file.

QUERY_PLAN_AS_HTML acts independently of the setting for the QUERY_PLAN option. In other words, if QUERY_PLAN_AS_HTML is ON, you get an HTML format query plan whether or not QUERY_PLAN is ON.

This feature is supported with newer versions of many commonly used browsers. Some browsers might experience problems with plans generated for very complicated queries.

QUERY_PLAN_AS_HTML_DIRECTORY option

Function	Specifies the directory into which Sybase IQ writes the HTML query plans.
Allowed values	String containing a directory path name
Default	'' (the empty string)

Scope

Can be set temporary, for an individual connection, or for the PUBLIC group. DBA authority is required to set the option. Takes effect immediately.

Description

When the QUERY_PLAN_AS_HTML option is turned ON and a directory is specified with the QUERY_PLAN_AS_HTML_DIRECTORY option, Sybase IQ writes the HTML query plans in the specified directory. This option provides additional security, as query plans can contain sensitive data. When the QUERY_PLAN_AS_HTML_DIRECTORY option is not used, the query plans are sent to the default directory (the *.iqmsg* file directory).

If the QUERY_PLAN_AS_HTML option is ON and QUERY_PLAN_AS_HTML_DIRECTORY is set to a directory that does not exist, Sybase IQ does not save the HTML query plan and no error is generated. In this case, the query continues to run and a message is logged to the IQ message file, so the DBA knows that the HTML query plan was not written. If the specified directory path or permissions on the directory are not correct, the message "Error opening HTML Query plan: *file-name*" is written in the *.iqmsg* file.

Example

Create the example directory */system1/users/DBA/html_plans* and set the correct permissions on the directory. Then set the options and run the query:

```
SET TEMPORARY OPTION QUERY_PLAN_AS_HTML = 'ON';
SET TEMPORARY OPTION QUERY_PLAN_AS_HTML_DIRECTORY = '/
system1/users/DBA/html_plans';
SELECT col1 FROM tab1;
```

The HTML query plan is written to a file in the specified directory */system1/users/DBA/html_plans*.

See also

"QUERY_PLAN_AS_HTML option" on page 140.

QUERY_ROWS_RETURNED_LIMIT option

Function

Sets the row threshold for rejecting queries based on estimated size of result set.

Allowed values

Any integer

Default

0

Scope

DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

If Sybase IQ receives a query that has an estimated number of result rows greater than the value of QUERY_ROWS_RETURNED_LIMIT, it rejects the query with this message:

```
Query rejected because it exceeds resource:
Query_Rows_Returned_Limit
```

If you set this option to zero (the default), there is no limit and no queries are ever rejected based on the number of rows in their output.

QUERY_TEMP_SPACE_LIMIT option

Function

Constrains the use of temporary IQ dbspace by user queries.

Allowed values

Any integer

Default

2000MB

Scope

DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

If Sybase IQ receives a query that requires a temporary result space larger than value of this option, it rejects the query with this message:

```
Query rejected because it exceeds total space resource
limit
```

If you set this option to zero, there is no limit and no queries are ever rejected based on their temporary dbspace requirements.

QUERY_TIMING option

Function

Determines whether or not to collect specific timing statistics.

Allowed values

ON, OFF

Default

OFF

Scope

DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description

This option controls the collection of timing statistics on subqueries and some other repetitive functions in the query engine. This parameter should normally be OFF (the default) because for very short correlated subqueries, timing every subquery execution can slow down a query.

QUOTED_IDENTIFIER option [TSQL]

Function	Controls the interpretation of strings that are enclosed in double quotes.
Allowed values	ON, OFF OFF for Open Client connections.
Default	ON
Description	QUOTED_IDENTIFIER controls whether strings enclosed in double quotes are interpreted as identifiers (ON) or as literal strings (OFF). This option is included for Transact-SQL compatibility. Sybase Central and Interactive SQL set QUOTED_IDENTIFER temporarily to ON if it is set to OFF. A message is displayed informing you of this change. The change is in effect only for the Sybase Central or Interactive SQL connection. The JDBC driver also turns QUOTED_IDENTIFIER to ON.
See also	Appendix A, "Compatibility with Other Sybase Databases.'

RECOVERY_TIME option

Function	Sets the maximum length of time, in minutes, that the database server takes to recover from system failure.
Allowed values	Integer, in minutes
Default	2
Scope	Can be set only for the PUBLIC group. Takes effect when the server is restarted.
Description	Use this option with the CHECKPOINT_TIME option to decide when checkpoints should be done. A heuristic measures the recovery time based on the operations since the last checkpoint. Thus, the recovery time is not exact.
See also	Chapter 10, "Transactions and Versioning" in the *Sybase IQ System Administration Guide*.

RETURN_DATE_TIME_AS_STRING option

Function	Controls how a date, time, or timestamp value is passed to the client application when queried.
Allowed values	ON, OFF

Default	OFF
Scope	Can be set as a temporary option only, for the duration of the current connection.
Description	RETURN_DATE_TIME_AS_STRING indicates whether date, time, and timestamp values are returned to applications as a date or time datatype or as a string. When this option is set to ON, the server converts the date, time, or timestamp value to a string before it is sent to the client in order to preserve the TIMESTAMP_FORMAT, DATE_FORMAT, or TIME_FORMAT option setting. Sybase Central and Interactive SQL automatically turn the RETURN_DATE_TIME_AS_STRING option ON.
See also	"DATE_FORMAT option" on page 63. "TIME_FORMAT option" on page 165. "TIMESTAMP_FORMAT option" on page 166.

ROW_COUNT option

Function	Limits the number of rows returned from a query.
Allowed values	Integer.
Default	0 (no limit on rows returned)
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	When this runtime option is set to a nonzero value, query processing stops after the specified number of rows. This option affects only statements with the keyword SELECT. It does not affect UPDATE and DELETE statements. The SELECT statement keywords FIRST and TOP also limit the number of rows returned from a query. FIRST returns the first row and is equivalent to setting ROW_COUNT equal to 1. TOP returns a specified number of rows and is the same as setting ROW_COUNT equal to the same number of rows. TOP has an upper limit of 32767, but ROW_COUNT has no upper limit. If both TOP and ROW_COUNT are set, the value of TOP takes precedence.
See also	"QUERY_ROWS_RETURNED_LIMIT option" on page 141. SELECT statement on page 632.

SCALE option

Function
: Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION, for queries on the Catalog Store only.

Allowed values
: Integer, with a maximum of 126.

Default
: 38

Description
: This option specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION, for queries on the Catalog Store.

  Multiplication, division, addition, subtraction, and aggregate functions may all have results that exceed the maximum precision.

See also
: "PRECISION option" on page 135.

  For queries on the IQ Store, see "MAX_CLIENT_NUMERIC_SCALE option."

SIGNIFICANTDIGITSFORDOUBLEEQUALITY option

Function
: Specifies the number of significant digits to the right of the decimal in exponential notation that are used in equality tests between two complex arithmetic expressions.

Allowed values
: 0 – 15

Default
: 0

Scope
: DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.

Description
: Because doubles are stored in binary (base 2) instead of decimal (base 10), this setting gives the approximate number of significant decimal digits used. If set to 0, all digits are used.

  For example, when the option is set to 12, the following numbers compare as equal. When set to 13, they do not:

  - 1.23456789012345
  - 1.23456789012389

  This option affects equality tests between two complex arithmetic expressions, not those done by the indexes.

SORT_PHASE1_HELPERS option

Function	Specifies the number of threads to be used in a sort.
Allowed values	Integer
Default	3
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	Use SORT_PHASE1_HELPERS to control the number of helper threads used by the sort object in phase1 (insertion). If set to 0, no helper threads are requested. Use this option for performance analysis and tuning. If you change this option, experiment to find the best value to increase performance, as choosing the wrong value might decrease performance. Sybase recommends using the default value for SORT_PHASE1_HELPERS. If your IQ temporary buffer cache is larger than 10GB, however, Sybase recommends setting the SORT_PHASE1_HELPERS option between 5 and 10.

SORT_PINNABLE_CACHE_PERCENT option

Function	Specifies the maximum percentage of currently available buffers a sort object tries to pin.
Allowed values	0 – 100
Default	20
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Description	For very large sorts, a larger value might help reduce the number of merge phases required by the sort. A larger number, however, might impact other users' sorts and hashes running on the system. If you change this option, experiment to find the best value to increase performance, as choosing the wrong value might decrease performance. Sybase recommends that you use the default value for SORT_PINNABLE_CACHE_PERCENT. This option is primarily for use by Sybase Technical Support. If you change the value of SORT_PINNABLE_CACHE_PERCENT, do so with extreme caution.

SQL_FLAGGER_ERROR_LEVEL option [TSQL]

Function — Controls the behavior in response to any SQL code that is not part of a specified set of SQL92.

Allowed values — E, I, F, or W

Default — W

Description — The SQL_FLAGGER_ERROR_LEVEL option flags as an error any SQL code that is not part of a specified set of SQL92. Allowed values and meanings are shown in Table 2-15.

Table 2-15: SQL_FLAGGER_ERROR_LEVEL values

Value	Action
E	Flag syntax that is not entry-level SQL92 syntax
I	Flag syntax that is not intermediate-level SQL92 syntax
F	Flag syntax that is not full-SQL92 syntax
W	Allow all supported syntax

SQL_FLAGGER_WARNING_LEVEL option [TSQL]

Function — Controls the behavior in response to any SQL that is not part of a specified set of SQL92.

Allowed values — E, I, F, or W

Default — W

Description — SQL_FLAGGER_WARNING_LEVEL flags as a warning any SQL that is not part of a specified set of SQL92. Allowed values of *level* and their meanings are shown in Table 2-16:

Table 2-16: SQL_FLAGGER_WARNING_LEVEL values

Value	Action
E	Flag syntax that is not entry-level SQL92 syntax
I	Flag syntax that is not intermediate-level SQL92 syntax
F	Flag syntax that is not full-SQL92 syntax
W	Allow all supported syntax

STATISTICS option [DBISQL]

Function — Controls whether execution times, optimization strategies, and other statistics display in the statistics window.

Allowed values — 0, 3, 4, 5, or 6

Default — 3

Description — When STATISTICS is set to 0, the statistics window is not displayed. Otherwise, the value represents the height of the statistics window in lines.

STRING_RTRUNCATION option [TSQL]

Function — Determines whether an error is raised when an INSERT or UPDATE truncates a CHAR or VARCHAR string.

Allowed values — ON, OFF

Default — OFF

Description — If the truncated characters consist only of spaces, no exception is raised. ON corresponds to SQL92 behavior. When STRING_TRUNCATION is OFF, the exception is not raised and the character string is silently truncated. If the option is ON and an error is raised, a ROLLBACK occurs.

SUBQUERY_PLACEMENT_PREFERENCE option

Function — Controls the placement of correlated subquery predicate operators within a query plan.

Allowed Values — -1 to 1

Default 0

Scope Can be set for any scope, any user, takes immediate effect.

Description For correlated subquery operators within a query, the IQ optimizer may have a choice of several different valid locations within that query's plan. SUBQUERY_PLACEMENT_PREFERENCE allows you to override the optimizer's cost-based decision when choosing the placement location. It does not override internal rules that determine whether a location is valid, and in some queries, there might be only one valid choice. If you set this option to a nonzero value, it affects every correlated subquery predicate in a query; it cannot be used to selectively modify the placement of one subquery out of several in a query.

This option is normally used for internal testing, and only experienced DBAs should use it. Table 2-17 describes the valid values for this option and their actions.

Table 2-17: SUBQUERY_PLACEMENT_PREFERENCE values

Value	Action
0	Let the optimizer choose.
1	Prefer a location high in the query plan, thereby delaying the execution of the subquery to as late as possible within the query.
-1	Prefer a location as low as possible in the query plan, thereby placing the execution of the subquery as early as possible within the query.

The default setting of this option is almost always appropriate. Occasionally, Sybase Technical Support might ask you to change this value.

SUPPRESS_TDS_DEBUGGING option

Function Determines whether TDS debugging information appears in the server window.

Allowed values ON, OFF

Default OFF

Description When the server is started with the -z option, debugging information appears in the server window, including debugging information about the TDS protocol.

The SUPPRESS_TDS_DEBUGGING option restricts the debugging information about TDS that appears in the server window. When this option is set to OFF (the default), TDS debugging information appears in the server window.

SWEEPER_THREADS_PERCENT option

Function	Specifies the percentage of Sybase IQ threads used to sweep out buffer caches
Allowed Values	1 – 40
Default	10
Scope	Can be set only for the PUBLIC group. DBA authority is required to set the option. You must shut down and restart the database server for the change to take effect.
Description	Sybase IQ uses a small percentage of its processing threads as sweeper threads. These sweeper threads clean out dirty pages in the main and temp buffer caches. In the IQ Monitor -cache report, the GDirty column shows the number of times the LRU buffer was grabbed in a “dirty” (modified) state. If GDirty is greater than 0 for more than a brief time, you might need to increase SWEEPER_THREADS_PERCENT or WASH_AREA_BUFFERS_PERCENT. The default setting of this option is almost always appropriate. Occasionally, Sybase Technical Support might ask you to increase this value.
See also	“WASH_AREA_BUFFERS_PERCENT option” on page 171. Chapter 6, “Monitoring and Tuning Performance” in the *Sybase IQ Performance and Tuning Guide*.

TDS_EMPTY_STRING_IS_NULL option [database]

Function	Controls whether empty strings are returned as NULL or a string containing one blank character for TDS connections.
Allowed values	ON, OFF
Default	OFF

Description

By default, TDS_EMPTY_STRING_IS_NULL is set to OFF and empty strings are returned as a string containing one blank character for TDS connections. When this option is set to ON, empty strings are returned as NULL strings for TDS connections. Non-TDS connections distinguish empty strings from NULL strings.

TEMP_CACHE_MEMORY_MB option

Function

Specifies the size of the temporary shared buffer cache, in MB.

Allowed values

1 to 4294967295 (2^{32} -1)

Default

12MB

Scope

Can be set only for the PUBLIC group. DBA authority is required to set the option. You must shut down and restart the database server for the change to take effect.

Description

This option sets the size of the temporary shared memory buffer cache for the database. Sybase recommends that you do not use this option; instead, use the -iqtc server option to set the temporary buffer cache size.

On 64-bit systems, you can allocate as much physical memory as you have to IQ buffer caches; however, for values greater than 4GB, you must use the server options -iqmc and -iqtc to set main and temporary buffer cache sizes. On 32-bit systems, the operating system limits the amount of memory you can allocate. See the *Sybase IQ Installation and Configuration Guide* for your platform for details.

In almost every case, the default temporary buffer cache size of 8MB is too low. For optimal performance, allocate as much memory as possible to the IQ main and temporary buffer caches. For example, if you have 4GB of shared memory on your machine available to Sybase IQ, you can split that amount between the main and temporary shared buffer caches.

If your IQ temporary buffer cache is larger than 10GB, Sybase also recommends increasing the SORT_PHASE1_HELPERS database option.

When setting the temp cache size, you must consider many factors, including total physical memory, swap space, memory for the main buffer cache, your indexes and query types, your mix of query and load processing, the number of concurrent users, and memory requirements of the operating system and other applications on the machine.

See also

Chapter 5, “Managing System Resources” in the *Sybase IQ Performance and Tuning Guide* for important information about setting buffer cache sizes.

"SORT_PHASE1_HELPERS option" on page 146.

"Server command-line switches" on page 8 in Chapter 1, "Running the Database Server," in the *Sybase IQ Utility Guide*.

TEMP_KB_PER_STRIPE option

Function	Defines the number of kilobytes (KB) to write to each dbspace before the disk striping algorithm moves to the next stripe for the IQ Temporary Store.
Allowed values	Integer greater than zero, in kilobytes
Default	1 (which rounds up to one page)
Scope	Can be set only for the PUBLIC group. DBA authority is required to set the option. Takes effect at the next checkpoint.
Description	TEMP_KB_PER_STRIPE lets you control the number of kilobytes written to each dbspace before the IQ disk striping algorithm moves to the next stripe for the IQ Temporary Store. The corresponding number of blocks is rounded up to a page boundary, so the actual amount written to each stripe might be slightly larger than requested. You can tune this option by measuring the time required to complete I/O intensive updates and adjusting the option value accordingly.
See also	Chapter 5, "Working with Database Objects" in the *Sybase IQ System Administration Guide*. Balancing I/O in Chapter 5, "Managing System Resources," in the *Sybase IQ Performance and Tuning Guide*.

TEMP_EXTRACT_APPEND option

Function	Specifies that any rows extracted by the data extraction facility are added to the end of an output file.
Allowed values	ON, OFF
Default	OFF
Scope	Can be set for an individual connection. Takes effect immediately.

Description

This option specifies that any rows extracted by the data extraction facility are added to the end of an output file. You create the output file in a directory where you have WRITE/EXECUTE permissions and you set WRITE permission on the directory and output file for the user name used to start Sybase IQ (for example, **sybase**). You can give permissions on the output file to other users as appropriate. The name of the output file is specified in the TEMP_EXTRACT_NAME1 option. The data extraction facility creates the output file, if the file does not already exist.

TEMP_EXTRACT_APPEND is not compatible with the TEMP_EXTRACT_SIZEn options. If you try to restrict the size of the extract append output file, Sybase IQ reports an error.

See also

For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

"TEMP_EXTRACT_NAMEn options" on page 155.

TEMP_EXTRACT_BINARY option

Function

In combination with the TEMP_EXTRACT_SWAP option, specifies the type of extraction performed by the data extraction facility.

Allowed values

ON, OFF

Default

OFF

Scope

Can be set for an individual connection. Takes effect immediately.

Description

Use this option with the TEMP_EXTRACT_SWAP option to specify the type of extraction performed by the data extraction facility.

Table 2-18: Extraction option settings for extraction type

Extraction type	TEMP_EXTRACT_ BINARY	TEMP_EXTRACT_ SWAP
binary	ON	OFF
binary/swap	ON	ON
ASCII	OFF	OFF

The default extraction type is ASCII.

See also

For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

“TEMP_EXTRACT_SWAP option” on page 163.

TEMP_EXTRACT_COLUMN_DELIMITER option

Function

Specifies the delimiter between columns in the output of the data extraction facility for an ASCII extraction.

Allowed values

String

Default

','

Scope

Can be set for an individual connection. Takes effect immediately.

Description

Use TEMP_EXTRACT_COLUMN_DELIMITER to specify the delimiter between columns in the output of the data extraction facility. In the case of an ASCII extraction, the default is to separate column values with commas. Strings are unquoted by default.

The delimiter must occupy 1 – 4 bytes, and must be valid in the collation order you are using, if you are using a multibyte collation order. Choose a delimiter that does not occur in any of the data output strings themselves.

If you set this option to the empty string '' for ASCII extractions, the extracted data is written in fixed-width ASCII with no column delimiter. Numeric and binary data types are right-justified on a field of *n* blanks, where *n* is the maximum number of bytes needed for any value of that type. Character data types are left-justified on a field of *n* blanks.

Note The minimum column width in a fixed-width ASCII extraction is 4 bytes to allow the string “NULL” for a NULL value. For example, if the extracted column is CHAR(2) and TEMP_EXTRACT_COLUMN_DELIMITER is set to the empty string '', there are two spaces after the extracted data.

See also

“TEMP_EXTRACT_QUOTE option” on page 159.

“TEMP_EXTRACT_QUOTES option” on page 159.

“TEMP_EXTRACT_ROW_DELIMITER option” on page 161.

“TEMP_EXTRACT_QUOTES_ALL option” on page 160.

For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, “Moving Data In and Out of Databases” in the *Sybase IQ System Administration Guide*.

TEMP_EXTRACT_DIRECTORY option

Function	Controls whether a user is allowed to use the data extraction facility.
Allowed values	string
Default	" (the empty string)
Scope	Can be set temporary, per user, or for the PUBLIC group. DBA authority is required to set the option. This option takes effect immediately.
Description	If the TEMP_EXTRACT_DIRECTORY option is set to the string FORBIDDEN (case insensitive) for a user, then that user is not allowed to perform data extracts. An attempt by this user to use the data extraction facility results in an error. If TEMP_EXTRACT_DIRECTORY is set to FORBIDDEN for the PUBLIC group, then no one can run data extraction. This option provides increased security and helps control disk management by restricting the creation of large data extraction files to the directories for which a user has write access.
See also	“TEMP_EXTRACT_NAMEn options” on page 155. For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, “Moving Data In and Out of Databases” in the *Sybase IQ System Administration Guide*.

TEMP_EXTRACT_NAMEn options

Function	Specifies the names of the output files or named pipes used by the data extraction facility. There are eight options: TEMP_EXTRACT_NAME1 through TEMP_EXTRACT_NAME8.
Allowed values	string
Default	" (the empty string)
Scope	Can be set for an individual connection. Takes effect immediately.
Description	TEMP_EXTRACT_NAME1 through TEMP_EXTRACT_NAME8 specify the names of the output files used by the data extraction facility. You must use these options sequentially. For example, TEMP_EXTRACT_NAME3 has no effect unless both the options TEMP_EXTRACT_NAME1 and TEMP_EXTRACT_NAME2 are already set.

The most important of these options is TEMP_EXTRACT_NAME1. If TEMP_EXTRACT_NAME1 is set to its default setting (the empty string ''), extraction is disabled and no output is redirected. To enable extraction, set TEMP_EXTRACT_NAME1 to a path name. Extract starts extracting into a file with that name. Choose a path name to a file that is not otherwise in use. Sybase recommends setting the TEMP_EXTRACT_NAME1 option as TEMPORARY.

You can also use TEMP_EXTRACT_NAME1 to specify the name of the output file, when the TEMP_EXTRACT_APPEND option is set ON. In this case, before you execute the SELECT statement, set WRITE permission for the user name used to start Sybase IQ (for example, **sybase**) on the directory or folder containing the named file and on the named file. In append mode, the data extraction facility adds extracted rows to the end of the file and does not overwrite the data that is already in the file. If the output file does not already exist, the data extraction facility creates the file.

Warning! If you choose the path name of an existing file and the TEMP_EXTRACT_APPEND option is set OFF (the default), the file contents are overwritten. This might be what you require if the file is for a weekly report, for example, but not if the file is one of your database files.

The options TEMP_EXTRACT_NAME2 through TEMP_EXTRACT_NAME8 can be used in addition to TEMP_EXTRACT_NAME1 to specify the names of multiple output files.

If you are extracting to a single disk file or a single named pipe, leave the options TEMP_EXTRACT_NAME2 through TEMP_EXTRACT_NAME8 and TEMP_EXTRACT_SIZE1 through TEMP_EXTRACT_SIZE8 at their default values.

When TEMP_EXTRACT_NAME1 is set, you cannot perform these operations:

- LOAD, DELETE, INSERT, or INSERT...LOCATION to a table that is the top table in a join
- SYNCHRONIZE JOIN INDEX (issued explicitly or executed as part of CREATE JOIN INDEX)
- INSERT...SELECT

Also note the following restrictions on the data extraction facility:

- Extract works only with data stored in the IQ Store.
- Extract does not work on system tables or cross database joins.

- Extract does not work with queries that use user-defined functions or system functions, except for the system functions suser_id() and suser_name().
- If you run DBISQL (Interactive SQL Java) with the -q (quiet mode) option and the data extraction commands are in a command file, you must first set and make permanent the DBISQL option "Show multiple result sets." If this option is not set, the output file is not created.

  To set the "Show multiple result sets" option, select Tools → Options in the DBISQL window, then check the box "Show multiple result sets" and click "Make permanent."

See also

"TEMP_EXTRACT_SIZEn options" on page 161.

"TEMP_EXTRACT_APPEND option" on page 152.

For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

TEMP_EXTRACT_NULL_AS_EMPTY option

Function

Controls the representation of null values in the output of the data extraction facility for an ASCII extraction.

Allowed values

ON, OFF

Default

OFF

Scope

Can be set for an individual connection. Takes effect immediately.

Description

TEMP_EXTRACT_NULL_AS_EMPTY controls the representation of null values in the output of the data extraction facility for ASCII extractions. When the TEMP_EXTRACT_NULL_AS_EMPTY option is set to ON, a null value is represented as '' (the empty string) for all data types.

The quotes shown above are not present in the extract output file. When the TEMP_EXTRACT_NULL_AS_EMPTY option is set to OFF, the string 'NULL' is used in all cases to represent a NULL value. OFF is the default value.

See also

For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

TEMP_EXTRACT_NULL_AS_ZERO option

Function
Controls the representation of null values in the output of the data extraction facility for an ASCII extraction.

Allowed values
ON, OFF

Default
OFF

Scope
Can be set for an individual connection. Takes effect immediately.

Description
TEMP_EXTRACT_NULL_AS_ZERO controls the representation of null values in the output of the data extraction facility for ASCII extractions. When TEMP_EXTRACT_NULL_AS_ZERO is set to ON, a null value is represented as follows:

- '0' for arithmetic type
- '' (the empty string) for the CHAR and VARCHAR character types
- '' (the empty string) for dates
- '' (the empty string) for times
- '' (the empty string) for timestamps

The quotes shown above are not present in the extract output file. When the TEMP_EXTRACT_NULL_AS_ZERO option is set to OFF, the string 'NULL' is used in all cases to represent a NULL value. OFF is the default value.

Note In Sybase IQ 12.5, an ASCII extract from a CHAR or VARCHAR column in a table always returns at least four characters to the output file. This is required if TEMP_EXTRACT_NULL_AS_ZERO is set to OFF, because Sybase IQ needs to write out the word NULL for any row in a column that has a null value. Reserving four spaces is not required if TEMP_EXTRACT_NULL_AS_ZERO is set to ON.

In Sybase IQ 12.6, if TEMP_EXTRACT_NULL_AS_ZERO is set to ON, the number of characters that an ASCII extract writes to a file for a CHAR or VARCHAR column equals the number of characters in the column, even if that number is less than four.

See also
For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, “Moving Data In and Out of Databases” in the *Sybase IQ System Administration Guide*.

TEMP_EXTRACT_QUOTE option

Function	Specifies the string to be used as the quote to enclose fields in the output of the data extraction facility for an ASCII extraction, when either the TEMP_EXTRACT_QUOTES option or the TEMP_EXTRACT_QUOTES_ALL option is set ON.
Allowed values	String
Default	" (the empty string)
Scope	Can be set for an individual connection. Takes effect immediately.
Description	This option specifies the string to be used as the quote to enclose fields in the output of the data extraction facility for an ASCII extraction, if the default value is not suitable. TEMP_EXTRACT_QUOTE is used with the TEMP_EXTRACT_QUOTES and TEMP_EXTRACT_QUOTES_ALL options.The quote string specified in the TEMP_EXTRACT_QUOTE option has the same restrictions as the row and column delimiters. The default for this option is the empty string, which Sybase IQ converts to the single quote mark. The string specified in the TEMP_EXTRACT_QUOTE option must occupy from 1 to a maximum of 4 bytes and must be valid in the collation order you are using, if you are using a multibyte collation order. Be sure to choose a string that does not occur in any of the data output strings themselves.
See also	For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, “Moving Data In and Out of Databases” in the *Sybase IQ System Administration Guide*. “TEMP_EXTRACT_COLUMN_DELIMITER option” on page 154. “TEMP_EXTRACT_QUOTES option” on page 159. “TEMP_EXTRACT_QUOTES_ALL option” on page 160. “TEMP_EXTRACT_ROW_DELIMITER option” on page 161.

TEMP_EXTRACT_QUOTES option

Function	Specifies that string fields are enclosed in quotes in the output of the data extraction facility for an ASCII extraction.
Allowed values	ON, OFF
Default	OFF
Scope	Can be set for an individual connection. Takes effect immediately.

Description — This option specifies that string fields are enclosed in quotes in the output of the data extraction facility for an ASCII extraction. The string used as the quote is specified in the TEMP_EXTRACT_QUOTE option, if the default is not suitable.

See also — For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

"TEMP_EXTRACT_COLUMN_DELIMITER option" on page 154.

"TEMP_EXTRACT_QUOTES option" on page 159.

"TEMP_EXTRACT_QUOTES_ALL option" on page 160.

"TEMP_EXTRACT_ROW_DELIMITER option" on page 161.

TEMP_EXTRACT_QUOTES_ALL option

Function — Specifies that all fields are enclosed in quotes in the output of the data extraction facility for an ASCII extraction.

Allowed values — ON, OFF

Default — OFF

Scope — Can be set for an individual connection. Takes effect immediately.

Description — TEMP_EXTRACT_QUOTES_ALL specifies that all fields are enclosed in quotes in the output of the data extraction facility for an ASCII extraction. The string used as the quote is specified in TEMP_EXTRACT_QUOTE if the default is not suitable.

See also — For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

"TEMP_EXTRACT_COLUMN_DELIMITER option" on page 154.

"TEMP_EXTRACT_QUOTES option" on page 159.

"TEMP_EXTRACT_QUOTES_ALL option" on page 160.

"TEMP_EXTRACT_ROW_DELIMITER option" on page 161.

TEMP_EXTRACT_ROW_DELIMITER option

Function
Specifies the delimiter between rows in the output of the data extraction facility for an ASCII extraction.

Allowed values
String

Default
" (the empty string)

Scope
Can be set for an individual connection. Takes effect immediately.

Description
TEMP_EXTRACT_ROW_DELIMITER specifies the delimiter between rows in the output of the data extraction facility. In the case of an ASCII extraction, the default is to end the row with a newline on UNIX platforms and with a carriage return/newline pair on Windows platforms.

The delimiter must occupy 1 – 4 bytes and must be valid in the collation order you are using, if you are using a multibyte collation order. Choose a delimiter that does not occur in any of the data output strings. The default for the TEMP_EXTRACT_ROW_DELIMITER option is the empty string. Sybase IQ converts the empty string default for this option to the newline on UNIX platforms and to the carriage return/newline pair on Windows platforms.

See also
For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

"TEMP_EXTRACT_COLUMN_DELIMITER option" on page 154.

"TEMP_EXTRACT_QUOTES option" on page 159.

"TEMP_EXTRACT_QUOTES_ALL option" on page 160.

"TEMP_EXTRACT_ROW_DELIMITER option" on page 161.

TEMP_EXTRACT_SIZEn options

Function
Specifies the maximum sizes of the corresponding output files used by the data extraction facility. There are eight options: TEMP_EXTRACT_SIZE1 through TEMP_EXTRACT_SIZE8.

Default
0

Scope
Can be set for an individual connection. Takes effect immediately.

Description

TEMP_EXTRACT_SIZE1 through TEMP_EXTRACT_SIZE8 are used to specify the maximum sizes of the corresponding output files used by the data extraction facility. TEMP_EXTRACT_SIZE1 specifies the maximum size of the output file specified by TEMP_EXTRACT_NAME1, TEMP_EXTRACT_SIZE2 specifies the maximum size of the output file specified by TEMP_EXTRACT_NAME2, and so on.

Note The default for the data extraction size options is 0. Sybase IQ converts this default to the values shown in the following table.

Device type	Size
Disk file	AIX and HP-UX: 0 – 64GB Sun Solaris & Linux: 0 – 512GB Windows: 0 – 128GB
Tape*	524288KB (0.5GB)
Other	9007199254740992KB (8192 Petabytes “unlimited”)

*Tape devices currently are not supported.

When large file systems, such as JFS2, support file size larger than the default value, set TEMP_EXTRACT_SIZEn to the value that the file system allows. For example, to support lTB set option:

```
TEMP_EXTRACT_SIZE1 = 1073741824 KB
```

If you are extracting to a single disk file or a single named pipe, leave the options TEMP_EXTRACT_NAME2 through TEMP_EXTRACT_NAME8 and TEMP_EXTRACT_SIZE1 through TEMP_EXTRACT_SIZE8 at their default values.

The TEMP_EXTRACT_SIZE*n* options are not compatible with TEMP_EXTRACT_APPEND. If you try to restrict the size of the extract append output file, Sybase IQ reports an error.

See also

For details on the data extraction facility and using the extraction options, see Data extraction options in Chapter 7, “Moving Data In and Out of Databases” in the *Sybase IQ System Administration Guide*.

“TEMP_EXTRACT_NAMEn options” on page 155.

TEMP_EXTRACT_SWAP option

Function
In combination with the TEMP_EXTRACT_BINARY option, specifies the type of extraction performed by the data extraction facility.

Allowed values
ON, OFF

Default
OFF

Scope
Can be set for an individual connection. Takes effect immediately.

Description
Use this option with the TEMP_EXTRACT_BINARY option to specify the type of extraction performed by the data extraction facility.

Table 2-19: Extraction option settings for extraction type

Extraction type	TEMP_EXTRACT_ BINARY	TEMP_EXTRACT_ SWAP
binary	ON	OFF
binary/swap	ON	ON
ASCII	OFF	OFF

The default extraction type is ASCII.

See also
For details on the data extraction facility and using the extraction options, see Data extraction options in the *Sybase IQ System Administration Guide*.

For details on the data extraction facility and using the extraction options, see "Data extraction options" on page 321 in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*

"TEMP_EXTRACT_BINARY option" on page 153

TEMP_RESERVED_DBSPACE_MB option

Function
Controls the amount of space Sybase IQ reserves in the Temporary IQ Store.

Allowed values
Integer greater than zero, in megabytes

Default
200; Sybase IQ actually reserves the minimum of 200MB and 50% of the size of the last dbspace

Scope
Can be set only for the PUBLIC group. DBA authority is required to set the option. Shut down and restart the database server for the change to take effect.

Description

TEMP_RESERVED_DBSPACE_MB lets you control the amount of space Sybase IQ sets aside in your Main IQ Store for certain small but critical data structures used during release savepoint, commit, and checkpoint operations. For a production database, set this value between 200MB and 1GB. The larger your IQ page size and number of concurrent connections, the more reserved space you need.

Sybase IQ reserves the minimum of 200MB and 50% of the size of the last dbspace, which helps DBAs avoid out-of-space conditions by reserving more space automatically.

See also

Reserving space to handle out-of-space conditions in Chapter 5, “Working with Database Objects” in the *Sybase IQ System Administration Guide*

TEMP_SPACE_LIMIT_CHECK option

Function

Checks for Catalog Store temporary space on a per connection basis.

Allowed values

ON, OFF (no limit checking occurs)

Default

OFF

Scope

Can be set only for the PUBLIC group. DBA authority required.

Description

When TEMP_SPACE_LIMIT_CHECK is ON, the database server checks the amount of Catalog Store temporary file space that a connection uses. If a connection requests more than its quota of temporary file space when this option is set to OFF, a fatal error can occur. When this option is set to ON, if a connection requests more than its quota of temporary file space, the request fails and the error “Temporary space limit exceeded” is returned.

Two factors are used to determine the temporary file quota for a connection: the maximum size of the temporary file, and the number of active database connections. The maximum size of the temporary file is the sum of the current size of the file and the amount of disk space available on the partition containing the file. When limit checking is turned on, the server checks a connection for exceeding its quota when the temporary file has grown to 80% or more of its maximum size, and the connection requests more temporary file space. Once this happens, any connection fails that uses more than the maximum temporary file space divided by the number of active connections.

Note This option is unrelated to IQ Temporary Store space. To constrain the growth of IQ temporary space, see "QUERY_TEMP_SPACE_LIMIT option" on page 142.

Example

A database is started with the temporary file on a drive with 100MB free and no other active files on the same drive. The available temporary file space is thus 100MB. The DBA issues:

```
SET OPTION PUBLIC.TEMP_SPACE_LIMIT_CHECK = 'ON'
```

As long as the temporary file stays below 80MB, the server behaves as it did before. Once the file reaches 80MB, the new behavior might occur. Assume that with 10 queries running, the temporary file needs to grow. When the server finds that one query is using more than 8MB of temporary file space, that query fails.

See also

You can obtain information about the space available for the temporary file using the sa_disk_free_space system procedure. For more information, see the *Adaptive Server Anywhere SQL Reference*.

TIME_FORMAT option

Function

Sets the format used for times retrieved from the database.

Allowed values

A string composed of the symbols HH, NN, MM, SS, separated by colons.

Default

'HH:NN:ss.SSS'

For Open Client and JDBC connections the default is set to HH:NN:SS.SSS.

Description

The format is a string using the following symbols:

- hh – Two-digit hours (24 hour clock).
- nn – Two-digit minutes.

- mm – Two-digit minutes if following a colon (as in 'hh:mm').
- ss[.s...s] – Two-digit seconds plus optional fraction.

Each symbol is substituted with the appropriate data for the date being formatted. Any format symbol that represents character rather than digit output can be in uppercase, which causes the substituted characters also to be in uppercase. For numbers, using mixed case in the format string suppresses leading zeros.

Multibyte characters are not supported in format strings. Only single-byte characters are allowed, even when the collation order of the database is a multibyte collation order like 932JPN.

See also

"DATE_FORMAT option" on page 63

"RETURN_DATE_TIME_AS_STRING option" on page 143.

TIMESTAMP_FORMAT option

Function

Sets the format used for timestamps retrieved from the database.

Allowed values

A string composed of the symbols listed below.

Default

'YYYY-MM-DD HH:NN:ss.SSS'

Description

The format is a string using the following symbols:

Table 2-20: TIMESTAMP_FORMAT string symbols

Symbol	Description
yy	2-digit year.
yyyy	4-digit year.
mm	2-digit month, or two digit minutes if following a colon (as in 'hh:mm').
mmm	3-character short form for name of the month of year
mmmm[m...]	Character long form for month name—as many characters as there are m's, until the number of m's specified exceeds the number of characters in the month's name.
dd	2-digit day of month.
ddd	3-character short form for name of the day of week.
dddd[d...]	Character long form for day name—as many characters as there are d's, until the number of d's specified exceeds the number of characters in the day's name.
hh	2-digit hours.
nn	2-digit minutes.
ss.SSS	Seconds (ss) and fractions of a second (SSS), up to six decimal places. Not all platforms support timestamps to a precision of six places.
aa	a.m. or p.m. (12-hour clock).
pp	p.m. if needed (12-hour clock.)

Each symbol is substituted with the appropriate data for the date being formatted. Any format symbol that represents character rather than digit output can be in uppercase, which causes the substituted characters also to be in uppercase. For numbers, using mixed case in the format string suppresses leading zeros.

Multibyte characters are not supported in format strings. Only single-byte characters are allowed, even when the collation order of the database is a multibyte collation order like 932JPN.

See also

"DATE_FORMAT option" on page 63

"RETURN_DATE_TIME_AS_STRING option" on page 143.

TRIM_PARTIAL_MBC option

Function	Allows automatic trimming of partial multibyte character data.
Allowed values	ON, OFF
Default	OFF
Scope	DBA permissions are not required to set this option. Can only be set for the PUBLIC group. Takes effect immediately.
Description	Provides consistent loading of data for collations that contain both single-byte and multibyte characters. When TRIM_PARTIAL_MBC is ON: • A partial multibyte character is replaced with a blank when loading into a CHAR column. • A partial multibyte character is truncated when loading into a VARCHAR column. When TRIM_PARTIAL_MBC is OFF, normal CONVERSION_ERROR semantics are in effect.
See also	"CONVERSION_ERROR option [TSQL]" on page 53.

TRUNCATE_WITH_AUTO_COMMIT option

Function	Speeds up TRUNCATE TABLE statements in the Catalog Store.
Allowed values	ON, OFF
Default	ON
Description	In the Catalog Store only, if TRUNCATE_WITH_AUTO_COMMIT is set to ON, then a COMMIT is executed both before and after the TRUNCATE TABLE statement is executed. The primary purpose of the option is to enable faster table truncation (delete of all rows). There are some cases where a fast TRUNCATE cannot be done: • If there are foreign keys either to or from the table • If the TRUNCATE TABLE statement is executed within an atomic statement
See also	TRUNCATE TABLE statement on page 658.

TRUNCATION_LENGTH option [DBISQL]

Function	Controls the truncation of wide columns for displays to fit on a screen.
Allowed values	Integer
Default	256
Description	When SELECT statement results are displayed on the screen, each column of output is limited to the width of the screen. The TRUNCATION_LENGTH option is used to reduce the width of wide columns so that more than one column fits on the screen. A value of 0 means that columns are not truncated. The default TRUNCATION_LENGTH is 256. For character-mode systems, this is an actual number of characters. For windowing systems, TRUNCATION_LENGTH is used to estimate an area of the screen to be used for display since proportional fonts are used.

TSQL_HEX_CONSTANT option [TSQL]

Function	Controls whether hexadecimal constants are treated as binary typed constants.
Allowed values	ON, OFF
Default	ON
Description	When set to ON, hexadecimal constants are treated as binary typed constants.

TSQL_VARIABLES option [TSQL]

Function	Controls whether the @ sign can be used as a prefix for Embedded SQL host variable names.
Allowed values	ON, OFF ON for Open Client and JDBC connections
Default	OFF
Description	When TSQL_VARIABLES is set to ON, you can use the @ sign instead of the colon as a prefix for host variable names in Embedded SQL. This is implemented primarily for the Open Server Gateway.

USER_RESOURCE_RESERVATION option

Function	Adjusts memory use for the number of current users.
Allowed values	Integer
Scope	DBA permissions are not required to set this option. Can be set temporary, for an individual connection, or for the PUBLIC group. Takes effect immediately.
Default	1
Description	Sybase IQ tracks the number of open cursors and allocates memory accordingly. In certain circumstances, you can use this option to adjust the minimum number of current cursors that Sybase IQ thinks is currently using the product, and allocate memory from the temporary cache more sparingly. Set this option only after careful analysis shows it is actually required. If you need to set this parameter, contact Sybase Technical Support with details.

VERIFY_PASSWORD_FUNCTION option

Function	Specifies a user-supplied authentication function that can be used to implement password rules. The function is called on a GRANT CONNECT TO *userid* IDENTIFIED BY *password* statement.
Allowed values	String
Scope	Can be set temporary, per user, or for the PUBLIC group. DBA authority is required to set the option. This option takes effect immediately.
Default	'' (the empty string). (No function is called on GRANT CONNECT.)
Description	When the VERIFY_PASSWORD_FUNCTION option value is set to a valid string, the statement GRANT CONNECT TO *userid* IDENTIFIED BY *password* calls the function specified by the option value. The option value requires the form *owner.function_name* to prevent users from overriding the function. The function takes two parameters: • *user_name* VARCHAR(128) • *new_pwd* VARCHAR(255)

It returns a value of type VARCHAR(255).

Note Perform an ALTER FUNCTION *function-name* SET HIDDEN on the function to ensure that a user cannot step through it using the procedure debugger.

If the VERIFY_PASSWORD_FUNCTION option is set, you cannot specify more than one userid and password with the GRANT CONNECT statement.

Example

For example, this statement creates a function that requires the password to be different from the user name:

```
CREATE FUNCTION DBA.f_verify_pwd
( user_name varchar(128),
new_pwd varchar(255) )
RETURNS varchar(255)
BEGIN
-- enforce password rules
IF new_pwd = user_name then
RETURN('Password cannot be the same as the user name' );
END IF;
-- return success
RETURN( NULL );
END;
ALTER FUNCTION DBA.f_verify_pwd set hidden;
GRANT EXECUTE on DBA.f_verify_pwd to PUBLIC;
SET OPTION PUBLIC.VERIFY_PASSWORD_FUNCTION =
'DBA.f_verify_pwd';
```

To turn the option off, set it to the empty string:

```
SET OPTION PUBLIC.VERIFY_PASSWORD_FUNCTION = ''
```

WASH_AREA_BUFFERS_PERCENT option

Function

Specifies the percentage of the buffer caches above the wash marker.

Allowed Values

1 – 100

Default

20

Scope

Can be set only for the PUBLIC group. DBA authority is required to set the option. Shut down and restart the database server to have the change take effect.

Description

Sybase IQ buffer caches are organized as a long MRU/LRU chain. The area above the wash marker is used to sweep out (that is, write) dirty pages to disk.

In the IQ Monitor -cache report, the Gdirty column shows the number of times the LRU buffer was grabbed in a "dirty" (modified) state. If GDirty is greater than 0 for more than a brief time, you might need to increase SWEEPER_THREADS_PERCENT or WASH_AREA_BUFFERS_PERCENT.

The default setting of this option is almost always appropriate. Occasionally, Sybase Technical Support might ask you to increase this value.

See also

Chapter 6, "Monitoring and Tuning Performance" in the *Sybase IQ Performance and Tuning Guide*

"SWEEPER_THREADS_PERCENT option" on page 150.

WAIT_FOR_COMMIT option

Function

Determines when foreign key integrity is checked as data is manipulated.

Allowed values

ON, OFF

Default

OFF

Scope

Can be set for an individual connection or the PUBLIC group. Takes effect immediately.

Description

If this option is set to ON, the database does not check foreign key integrity until the next COMMIT statement. Otherwise, all foreign keys not created with the CHECK ON COMMIT option are checked as they are inserted, updated, or deleted.

CHAPTER 3

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 employee
```

The following statements are equivalent to the one above:

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

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 3-1 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 3-1 lists the SQL reserved words in Sybase IQ.

Table 3-1: 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.

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 174.

For information on the QUOTED_IDENTIFIER option, see “The quoted_identifier option” on page 187.

For additional restrictions on server and database names, see “Server command-line switches” on page 8 in Chapter 1, “Running the Database Server” of the *Sybase IQ 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 179.

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 on page 553. 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:

```
employee.name
address
```

```
"date hired"
"salary"."date paid"
```

For more information, see “Identifiers” on page 177.

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 dept_name, COUNT(*), 'out of',
(SELECT COUNT(*) FROM employee)
FROM department AS D, employee AS E
WHERE D.dept_id = E.dept_id
GROUP BY dept_name;
```

For other uses of subqueries, see “Subqueries in search conditions” on page 191.

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 189.

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. You must turn off the PERCENT_AS_COMMENT option to use the '%' operator.

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.

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

Bit 1	Bit 2	Bit 1 & Bit 2
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
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** You can use the % operator in Sybase IQ only if the PERCENT_AS_COMMENT option is set to OFF. 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 make the order of operation explicit using parentheses, 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 218 and "Search conditions" on page 189.

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

For information on the IF statement, see IF statement on page 564.

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 "DBA".Product
```

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 "DBA".Product
```

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 3-2 and Table 3-3 describe the compatibility of expressions and constants between Adaptive Server Enterprise 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 3-2, expr represents an expression, and op represents an operator.

Table 3-2: 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 3-3: 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 3-1 on page 175.

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 employee
WHERE emp_lname = "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 name, year_of_birth
FROM employees
WHERE year_of_birth <= ALL (SELECT year_of_birth 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 179.

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 198.

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 218.

Subqueries form an important class of expression that is used in many search conditions. For more information, see "Subqueries in search conditions" on page 191.

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 3-4 lists the comparison operators available in Sybase IQ.

Table 3-4: 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 name, year_of_birth
FROM employees
WHERE year_of_birth <= ALL (SELECT MIN(year_of_birth)
```

```
    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

---

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 an option when creating the database.
- **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.

For more information, see "Comparison conditions" on page 190.

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:

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

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. 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.

The keyword SOME can be used instead of ANY.

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." In Sybase IQ 12.6 and prior versions, "A", "B", and "C" all had to be value expressions or columns.

Now either "B" or "C" or both "B" and "C" can be subqueries. "A" must still 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.

Leading substring searches are supported for LIKE conditions as follows:

- col LIKE 'ABC% DEF%' accelerates the entire LIKE
- col LIKE '%ABC% DEF%' accelerates only the DEF% word
- col LIKE 'ABC% DEF%' accelerates only the ABC word

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 *Sybase IQ System Administration Guide*.

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 either a CHAR or VARCHAR 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 column's word index.

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*

The combined condition is TRUE if both conditions are TRUE, FALSE if either condition is FALSE, and UNKNOWN otherwise.

Conditions are combined using OR as follows:

condition1 **OR** *condition2*

The combined condition is TRUE if either condition is TRUE, FALSE if both conditions are FALSE, and UNKNOWN otherwise.

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

OR	TRUE	FALSE	UNKNOWN
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 the 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 will be before than 1994/06/30:

```
SELECT  ship_date
FROM  sales_order_items
WHERE ( ship_date < '1994/06/30', 1.5 )
ORDER BY ship_date DESC
```

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

```
SELECT *
FROM customer c, sales_order o
WHERE (c.unpaid_balance > 10000.0, 0.5)
   AND c.id o.cust_id
```

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 199.

```
SELECT *
FROM customer c, sales_order o
WHERE (c.unpaid_balance > 10000.0, 's: 0.5')
   AND c.id = o.cust_id
```

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” on page 88 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 customer c, sales_order o
WHERE (c.unpaid_balance > 8000.0, 'S:3.00, I:+2')
   AND c.id = o.cust_id
```

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 customer c, sales_order o
WHERE (c.unpaid_balance > 3000.0, 'i:-2, s:37.500')
   AND c.id = o.cust_id
```

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 four distinct phases of execution where conditions can be evaluated. These phases are named 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 customer c, sales_order o
WHERE (c.unpaid_balance > 10000.0, 'E:D, I:1')
    AND c.id = o.cust_id AND o.order_price > 5000.0
```

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 customer c, sales_order o
WHERE (c.unpaid_balance > 10000.0, 'U: 3.25,  E: D')
AND c.id = o.cust_id AND o.order_price > 5000.0
```

Compatibility

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

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

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 179

CURRENT DATE special value

Function: The current year, month and day.

Data type: DATE

See also: "Expressions" on page 179

"Date and time data types" on page 234

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 179

Sybase IQ Troubleshooting and Recovery Guide

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 179

“Date and time data types” on page 234

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 179

“Date and time data types” on page 234

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 179

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 206

"CURRENT TIMESTAMP special value" on page 206

CREATE TABLE statement on page 499

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 179

Sybase IQ Troubleshooting and Recovery Guide

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 179

Sybase IQ Troubleshooting and Recovery Guide

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 234

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 179

"CURRENT USER special value" on page 206

"LAST USER special value" on page 206

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 is different in 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 3-5 lists the global variables available in Sybase IQ.

Table 3-5: 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 3-6 includes all Adaptive Server Enterprise global variables supported in Sybase IQ. Adaptive Server Enterprise global variables not supported by Sybase IQ are not included in the list. In contrast to Table 3-5, 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 3-6: 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.

Global variable	Returns
@@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.
@@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.

Global variable	Returns
@@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.
@@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 on the current version of Adaptive Server Enterprise or Sybase IQ.

Comments

Comments are used 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, if the PERCENT_AS_COMMENT option is set to ON. Using % as a comment indicator is not recommended.

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 "dba".employee
WHERE dept_id = 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 189 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

Condition	Truth value	Selected?
NOT (Salary <> NULL)	UNKNOWN	NO
Salary = 1000	UNKNOWN	NO
Salary IS NULL	TRUE	YES
Salary IS NOT NULL	FALSE	NO
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 5, "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 189 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 4

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.) For more information, see *Large Objects Management in Sybase IQ*.

Storage sizes

Table 4-1 lists the storage size of character data.

Table 4-1: 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.

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 Adaptive Server Anywhere. For more information on the Sybase IQ data type LONG VARCHAR, 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

Adaptive Server 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 111 and the SET OPTION statement on page 647.

Table 4-2 lists the storage required for a decimal number.

Table 4-2: 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 1.79769313486231570e+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.

You can tune the behavior of the FLOAT data type for compatibility with Adaptive Server Enterprise using the “FLOAT_AS_DOUBLE option [TSQL]” on page 77.

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*.
- If a join column is a REAL data type, you must set FLOAT_AS_DOUBLE to OFF when creating join indexes, or an error occurs. Issues may also result when using inexact numerics for join columns.

- 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
Adaptive Server Anywhere	3	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.) For more information, see *Large Objects Management in Sybase IQ*.

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 WHERE bnot = 0x00;
```

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

If the input value does not include the “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

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 4-3 lists the storage size of binary data.

Table 4-3: 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 261.

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 types is the same in Adaptive Server Anywhere and Sybase IQ, but is different in Adaptive Server Enterprise. Table 4-4 shows the differences.

Table 4-4: Treatment of trailing zeros

Data type	ASA and IQ	ASE
BINARY NOT NULL	padded	padded
BINARY NULL	padded	truncated
VARBINARY NOT NULL	not padded, not truncated	truncated
VARBINARY NULL	not padded, not truncated	truncated

Adaptive Server Enterprise, Adaptive Server 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 OFF.

Bit operations on binary type data are not supported by ASE. ASA only supports bit operations against the first four bytes of binary type data. Sybase IQ supports bit operations against the first 8 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 Adaptive Server Anywhere. Not supported by Adaptive Server Enterprise.
- **Backwards compatibility** In databases created before version 12.7, the STRTOUUID, UUIDTOSTR, and NEWID functions were supported through CIS functional compensation. The STRTOUUID, UUIDTOSTR, and NEWID functions are native Sybase IQ functions in Sybase IQ 12.7.

See also

For more information related to UNIQUEIDENTIFIER see also:

- “NEWID function [Miscellaneous]” on page 329
- “UUIDTOSTR function [String]” on page 381
- “STRTOUUID function [String]” on page 372

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 non-zero 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 242.

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 242.

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 datatime, 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', '7911-12-30 00:00:44'
```

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

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

1, 'example', '7911-12-30 23:59:59', '7911-12-30
06:03:44'
```

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 on page 647.

For information on functions dealing with dates and times, see "Date and time data types" on page 234. 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 multibye collation order like SJIS2.

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

```
SELECT DATEFORMAT ( start_date, 'yy?') FROM employee;
```

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

```
SELECT DATEFORMAT (start_date, 'yy') + '?' FROM
employee;
```

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 on page 647). For example, the following statement sets the DATE_ORDER option to ‘DMY’:

```
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. The value can still be changed using the SET OPTION statement.

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. For more information, see the “NEAREST_CENTURY option [TSQL]”.

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. For a full description of the syntax, see CREATE DOMAIN statement on page 456.

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 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 261.

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 a string containing only a time value (no date) is converted to a date/time data type, Sybase IQ and Adaptive Server Enterprise both use a default date of January 1, 1900. Adaptive Server 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 Adaptive Server 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 *Sybase IQ System Administration Guide*.

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 ASE 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
```

Year 2000 compliance

Description

The problem of handling dates, especially year values beyond the year 2000, has been a significant issue for the computer industry.

This section examines year 2000 compliance by Sybase IQ. It illustrates how Sybase IQ handles date values internally, and how it handles ambiguous date information such as the conversion of a 2-digit year string value.

Users of Sybase Anywhere and its predecessors can be assured that dates are handled and stored internally in a manner not adversely effected by the transition from the 20th century to the 21st century.

Consider the following measurements of Sybase IQ year 2000 compliance:

- It always returns correct values for any legal arithmetic and logical operations on dates, regardless of whether the calculated values span different centuries.
- At all times, the internal storage of dates explicitly includes the century portion of a year value.
- The operation is unaffected by any return value, including the current date.

- Date values can always be output in full century format.

Many of the date-related topics summarized in this section are explained in greater detail in other parts of the documentation.

How dates and times are stored

Dates containing year values are used internally and stored in Sybase IQ databases using the data types listed in Table 4-5.

Table 4-5: Storage of dates containing year values

Data type	Contains	Stored in	Range of possible values
DATE	Calendar date (year, month, day)	4 bytes	0001-01-01 to 9999-12-31
TIMESTAMP	Time stamp (year, month, day, hour minute, second, and fraction of second accurate to 6 decimal places)	8 bytes	0001-01-01 00:00:00.000000 to 9999-12-31 23:59:59.999999
TIME	Time of day (hour minute, second, and fraction of second accurate to 6 decimal places) since midnight	8 bytes	00:00:00.000000 to 23:59:59.999999

For more information on Sybase IQ date and time data types see “Date and time data types” on page 234.

Sending and retrieving date values

Date values are stored within Sybase IQ as either a DATE or TIMESTAMP data type. Time values are stored as a TIME or TIMESTAMP data type. They are passed to and retrieved from it using either of three methods:

- As a string, using any Sybase IQ programming interface.
- As a TIMESTAMP structure using ODBC.
- As a SQLDATETIME structure using Embedded SQL.

A string containing a date value is considered unambiguous and is automatically converted to a DATE or TIMESTAMP data type without potential for misinterpretation if it is passed using the following format: *yyyy-mm-dd* (the dash separator is one of several characters that are permitted).

To use date formats other than *yyyy-mm-dd* set the DATE_FORMAT database option (see SET OPTION statement on page 647).

Similarly, a string containing a time value is considered unambiguous and is automatically converted to a TIME or TIMESTAMP data type without potential for misinterpretation if it is passed using the following format: *hh:mm:ss.ssssss*.

For more information on unambiguous date formats, see the section "Using unambiguous dates and times" on page 238.

For more information on the ODBC TIMESTAMP structure see the Microsoft Open Database Connectivity SDK, or the section "Sending dates and times to the database" on page 236.

Used in the development of C programs, an Embedded SQL SQLDATETIME structure's year value is a 16-bit signed integer.

Leap years

The year 2000 is also a leap year, with an additional day in the month of February. Sybase IQ uses a globally accepted algorithm for identifying leap years. A year is considered a leap year if it is divisible by four, unless the year is a century date (such as the year 1900), in which case it is a leap year if it is divisible by 400.

Sybase IQ handles all leap years correctly. For example, this SQL statement results in a return value of "Tuesday":

```
SELECT DAYNAME('2000-02-29');
```

It accepts Feb 29, 2000—a leap year—as a date and using this date determines the day of the week on which that date occurs.

However, the following statement is rejected:

```
SELECT DAYNAME('2001-02-29');
```

This statement results in an error (cannot convert'2001-02-29' to a date) because Feb 29 does not exist in the year 2001.

Ambiguous string to date conversions

Sybase IQ automatically converts a string into a date when a date value is expected, even if the year is represented in the string by only 2 digits.

If the century portion of a year value is omitted, the conversion method is determined by the NEAREST_CENTURY database option.

The NEAREST_CENTURY database option is a numeric value that acts as a break point between *19yy* date values and *20yy* date values.

Two-digit years less than the NEAREST_CENTURY value are converted to *20yy*, while years greater than or equal to the value are converted to *19yy*.

If this option is not set, the default setting of 50 is assumed (0 to 49 are in the 21st century, 50 to 99 are in the 20th century).

This NEAREST_CENTURY option was introduced in Anywhere Release 5.5.

Ambiguous date conversion example

The following statement creates a table that can be used to illustrate the conversion of ambiguous date information in Sybase IQ.

```
CREATE TABLE T1 (C1 DATE);
```

The table T1 contains one column, C1, of the type DATE.

The following statement inserts a date value into the column C1. It automatically converts a string that contains an ambiguous year value, one with 2 digits representing the year but nothing to indicate the century.

```
INSERT INTO T1 VALUES('00-01-01');
```

By default, the NEAREST_CENTURY option is set to 50, thus Sybase IQ converts the string into the date 2000-01-01. Verify the result of this insert by entering:

```
SELECT * FROM T1;
```

Change the NEAREST_CENTURY option using the following statement to alter the conversion process:

```
SET OPTION NEAREST_CENTURY = 0;
```

When NEAREST_CENTURY is set to 0, executing the previous insert using the same statement creates a different date value:

```
INSERT INTO T1 VALUES('00-01-01');
```

The above statement now results in the insertion of the date 1900-01-01. Use the following statement to verify the results:

```
SELECT * FROM T1;
```

Date-to-string conversions

Sybase IQ provides several functions for converting date and time values into a wide variety of strings and other expressions. You can, in converting a date value into a string, reduce the year portion into a two-digit number representing the year, thereby losing the century portion of the date.

Wrong century values

Consider the following statement, which incorrectly converts a string representing Jan 1, 1900 into a string representing Jan 1, 2000 even though no database error occurs.

```
SELECT DATEFORMAT (DATEFORMAT('1900-01-01',
'Mmm dd/yy' ), 'yyyy-Mmm-dd' ) AS Wrong_year;
```

Although the unambiguous date string 1900-01-01 is automatically and correctly converted by Sybase IQ into a date value, the 'Mmm dd/yy' formatting of the inner, or nested DATEFORMAT function drops the century portion of the date when it is converted back to a string and passed to the outer DATEFORMAT function.

Because the database option NEAREST_CENTURY, in this case, is set to 50 the outer DATEFORMAT function converts the string representing a date with a 2-digit year value into a year in the 21st century.

For more information about ambiguous string conversions, see the section "Ambiguous string to date conversions" above.

For more information on date and time functions, see "Date and time data types" on page 234.

CHAPTER 5

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.

Remember the following 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, Adaptive Server 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 Adaptive Server 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 252.

Table 5-1 lists the aggregate functions and their parameters.

Table 5-1: Aggregate functions

Aggregate function	Parameters
AVG	([DISTINCT] { *column-name* | *numeric-expr* })
COUNT	(*)
COUNT	([DISTINCT] { *column-name* | *numeric-expr* })
MAX	([DISTINCT] { *column-name* | *numeric-expr* })
MIN	([DISTINCT] { *column-name* | *numeric-expr* })
STDDEV	([ALL] *expression*)
SUM	([DISTINCT] { *column-name* | *numeric-expr* })
VARIANCE	([ALL] *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 the OLAP functions, see Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*.

Analytical functions

Function

Analytical functions include the following:

- Simple aggregates — AVG, COUNT, MAX, MIN, and SUM, STDDEV and VARIANCE

  Note All simple aggregates, except the Grouping() function, can be used in 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:

```
olap_function() 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:

```
OLAP_FUNCTION() 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 12.7.

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 5-2 lists the analytical functions and their parameters. Unlike aggregate functions in Table 5-1, you cannot specify DISTINCT in window functions.

Table 5-2: 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*)
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.

For more information about using the OLAP functions, see Chapter 4, “Using OLAP” in the *Sybase IQ Performance and Tuning Guide*

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 5-3 and Table 5-4 list the date and time functions and their parameters.

Syntax 1

Table 5-3: Date and time functions

Date and time functions	Parameters
DATE	(*expression*)
DATEFORMAT	(*datetime-expr*, *string-expr*)
DATENAME	(*date-part*, *date-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*)
WEEKS	(*date-expr*, *integer-expr*)
YEAR	(*date-expr*)
YEARS	(*date-expr*)

Date and time functions	Parameters
YEARS	(*date-expr*, *date-expr*)
YEARS	(*date-expr*, *integer-expr*)
YMD	(*year-num*, *month-num*, *day-num*)

Syntax 2

Table 5-4: 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 5-3 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 5-4 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 Adaptive Server 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 Adaptive Server Anywhere. The following statement is handled exclusively by Adaptive Server 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. Adaptive Server Anywhere does not have a single starting date. The following statements, the first processed by Adaptive Server Anywhere, the second by Sybase IQ, both return the answer 12:

```
SELECT MONTHS('0001/01/01');
SELECT MONTHS('0001/01/01') FROM iq_dummy;
```

On the other hand, consider the following statements:

```
SELECT DAYS('0001/01/01');
SELECT DAYS('0001/01/01') FROM iq_dummy;
```

,The first, processed by Adaptive Server Anywhere, yields the value 307, but the second, processed by Sybase IQ, yields 166.

For the most consistent results, therefore, you should 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 with date or time functions, thus insuring processing by Sybase IQ, and consistent results.

Date parts

Many of the date functions use dates built from date parts. Table 5-5 displays allowed values of *date-part*.

Table 5-5: 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, set the following option:

```
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" on page 63.

Compatibility

For compatibility with Adaptive Server Enterprise, use the Transact-SQL date and time functions.

Data type conversion functions

Function

Data type conversion functions convert arguments from one data type to another.

Table 5-6 lists the data type conversion functions and their parameters.

Table 5-6: Date type conversion functions

Data type conversion function	Parameters
BIGINTTOHEX	(*integer-expression*)
CAST	(*expression* AS *datatype*)
CONVERT	(*datatype*, *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 256. The STRING function, which converts expressions to a string, is discussed in the section “String functions” on page 264.

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 241.

HTTP functions

Function

HTTP functions facilitate the handling of HTTP requests within Web services.

Table 5-7 lists all HTTP functions and their parameters.

Table 5-7: HTTP functions

HTTP function	Parameters
HTML_DECODE	(*string*)
HTML_ENCODE	(*string*)

HTTP function	Parameters
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 Adaptive Server 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 Adaptive Server 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 5-8 lists numeric functions and their parameters.

Table 5-8: 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*)

Numeric function	Parameters
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 8-1 on page 676.

Sybase IQ queries containing one or more of such 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 5-9 lists string functions and their parameters.

Table 5-9: 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*)
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_expr* [*collation-name*])
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*)
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 5-10 lists the system functions and their parameters.

Table 5-10: 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	({ NULL | *connection-id* })
NEXT_DATABASE	({ NULL | *database-id* })
OBJECT_ID	(*object-name*)
OBJECT_NAME	(*object-id* [, *database-id*])
PROPERTY	({ *property-number* | *property-name* })
PROPERTY_DESCRIPTION	({ *property-number* | *property-name* })
PROPERTY_NAME	(*property-number*)
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. For this reason, when queries to Sybase IQ tables include system functions, their performance is reduced.

Compatibility

Table 5-11 shows the Adaptive Server Enterprise system functions and their status in Sybase IQ:

Table 5-11: 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 of the connection properties, see the section “Database properties” in the chapter “Database Performance and Connection Properties” in the *Adaptive Server Anywhere Database Administration Guide*.

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 is displayed 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 of the server properties, see the section “Database properties” in the chapter “Database Performance and Connection Properties” in the *Adaptive Server Anywhere Database Administration Guide*.

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 the section "Database properties" in the chapter "Database Performance and Connection Properties" in the *Adaptive Server Anywhere Database Administration Guide*

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 Adaptive Server 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 ASA 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 ASA 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 8, "Using Procedures and Batches" in the *Sybase IQ System Administration Guide*.

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 "A Java Seminar" in the chapter "Introduction to Java in the Database" in the *Adaptive Server Anywhere Programming Guide*.

Miscellaneous functions

Function

Miscellaneous functions perform operations on arithmetic, string, or date/time expressions, including the return values of other functions.

Table 5-12 lists the miscellaneous functions and their parameters.

Table 5-12: 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 does not support these miscellaneous 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 274

"ATAN function [Numeric]" on page 274

"ATAN2 function [Numeric]" on page 275

"COS function [Numeric]" on page 287

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 273

"ATAN function [Numeric]" on page 274

"ATAN2 function [Numeric]" on page 275

"SIN function [Numeric]" on page 361

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 273
"ASIN function [Numeric]" on page 274
"ATAN2 function [Numeric]" on page 275
"TAN function [Numeric]" on page 376

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 273
"ASIN function [Numeric]" on page 274
"ATAN function [Numeric]" on page 274
"TAN function [Numeric]" on page 376

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 employee
```

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 287

"SUM function [Aggregate]" on page 374

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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]" on page 53
"HEXTOBIGINT function [Data type conversion]" on page 305
"HEXTOINT function [Data type conversion]" on page 306
"INTTOHEX function [Data type conversion]" on page 314

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 Adaptive Server Anywhere or Adaptive Server Enterprise

See also
"OCTET_LENGTH function [String]" on page 338

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 280

"DATALENGTH function [System]" on page 288

"LENGTH function [String]" on page 320

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 customer
```

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 compatibl.
- **Sybase** Not supported in Adaptive Server Enterprise

See also

"CONVERT function [Data type conversion]" on page 284

CEIL function [Numeric]

Function

Returns the 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 279

Chapter 11, "International Languages and Character Sets" in the *Sybase IQ System Administration Guide*

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 303

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 277

CHARINDEX function [String]

Function

Returns the position of the first occurrence of one string in another.

Syntax

CHARINDEX (*string-expression1*, *string-expression2*)

Parameters

string-expression1 The string you are searching for. This string is limited to 255 bytes.

string-expression2 The string to be searched.

Example

The statement:

```
SELECT emp_lname, emp_fname
FROM employee
WHERE CHARINDEX('K', emp_lname ) = 1
```

returns the following values:

emp_lname	emp_fname
Klobucher	James
Kuo	Felicia
Kelly	Moira

The position of the first character in the string being searched is 1.

Usage

If the string being searched contains more than one instance of the other string, CHARINDEX returns the position of the first instance.

If the string being searched does not contain the other string, CHARINDEX returns 0.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

See also

"SUBSTRING function [String]" on page 373

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** Not supported by 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 ( 'CUSTOMER', '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 288

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 customer 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 asiqdemo 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 296

"OBJECT_ID function [System]" on page 337

"SYSCOLUMN system table" on page 694

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*])

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 269

"PROPERTY_NUMBER function [System]" on page 347

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 5-13 lists the meanings of the values of the *format-style* argument.

Table 5-13: 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
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
```

Standards and compatibility

- **SQL92** Vendor extension.
- **Sybase** Compatible with Adaptive Server Enterprise and Adaptive Server Anywhere, except for format style 365, which is an IQ only extension.

See also

"CAST function [Data type conversion]" on page 278

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 273
"COT function [Numeric]" on page 287
"SIN function [Numeric]" on page 361
"TAN function [Numeric]" on page 376

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 287
"SIN function [Numeric]" on page 361
"TAN function [Numeric]" on page 376

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 employee
GROUP BY city
```

Standards and compatibility

- **SQL92** SQL92 compatible.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

"AVG function [Aggregate]" on page 275

"SUM function [Aggregate]" on page 374

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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 5-14 lists the return values of DATALENGTH.

Table 5-14: 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 customer
```

Standards and compatibility

- **SQL92** Vendor extension.
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ.

See also

"CHAR_LENGTH function [String]" on page 280

"COL_LENGTH function [System]" on page 282

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 259.

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.

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 259.

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.

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.

For information on date format descriptions, see "DATE_FORMAT option" on page 63.

Example

The following statement returns string values like "Jan 01, 1989":

```
SELECT DATEFORMAT( start_date, 'Mmm dd, yyyy' ) from
employee;
```

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 not necessary.

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 SJIS2.

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

```
SELECT DATEFORMAT ( start_date, 'yy?') FROM employee;
```

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

```
SELECT DATEFORMAT (start_date, 'yy') + '?' FROM
employee;
```

Standards and compatibility

- **SQL92** Vendor extension.
- **Sybase** Compatible with Adaptive Server Anywhere.

Year 2000 compliance

Do not use the DATEFORMAT function to produce a string with the year value represented by only two digits. This can cause problems with year 2000 compliance even though no error has occurred.

For more information on year 2000 compliance, see "Year 2000 compliance" on page 244.

See also

"DATE_FORMAT option" on page 63

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 259.

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 259.

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.

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

This statement:

```
SELECT DATETIME( '1998-09-09 12:12:12.000' ) FROM
iq_dummy
```

returns a timestamp with value 1998-09-09 12:12:12.000:

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** Not supported by 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 278.

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*])

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 asiqdemo is the only running database:

```
SELECT DB_ID( 'asiqdemo' ) 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 296

"OBJECT_ID function [System]" on page 337

DB_NAME function [System]

Function

Returns the database name.

Syntax

DB_NAME ([*database-id*])

Parameters

database-id The ID of the database. The *database-id* must be a numeric expression.

Example

The following statement returns the database name asiqdemo, 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 282

"DB_ID function [System]" on page 296

"OBJECT_NAME function [System]" on page 337

DB_PROPERTY function [System]

Function

Returns the value of the given property.

Syntax

DB_PROPERTY ({ *property-id* | *property-name* } [, { *database-id* | *database-name* }])

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 270

"DB_ID function [System]" on page 296

"DB_NAME function [System]" on page 296

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 Adaptive Server Anywhere.

See also

"Analytical functions" on page 252

"RANK function [Analytical]" on page 349

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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 364

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" on page 63 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.

EVENT_CONDITION function [System]

Function

To specify when an event handler is triggered.

Syntax

EVENT_CONDITION (*condition-name*)

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 5-15 lists the conditions and the events for which they are valid.

Table 5-15: 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 ASTMP 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" on page 458

EVENT_CONDITION_NAME function [System]

Function

Can be used to list the possible parameters for EVENT_CONDITION.

Syntax

EVENT_CONDITION_NAME (*integer*)

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" on page 458

EVENT_PARAMETER function [System]

Function

Provides context information for event handlers.

Syntax

EVENT_PARAMETER (*context-name*)

context-name:
'ConnectionID'
| **'User'**
| **'EventName'**
| **'Executions'**
| **'NumActive'**
| **'TableName'**
| *condition-name*

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 300

CREATE EVENT statement on page 458

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 3269017.372472109:

```
SELECT EXP( 15 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Vendor extension.
- **Sybase** Compatible with Adaptive Server Enterprise.

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 279

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.

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 on page 632

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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]” on page 53

“BIGINTTOHEX function [Data type conversion]” on page 276

“HEXTOINT function [Data type conversion]” on page 306

“INTTOHEX function [Data type conversion]” on page 314

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 set to 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 enabled (the value 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" on page 44

"CONVERSION_ERROR option [TSQL]" on page 53

"INTTOHEX function [Data type conversion]" on page 314

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 278.

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*)

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
"	&quot;
'	&#39;
&amp;	&
&lt;	<
&gt;	>
&#xnn;	character nn

Standards and compatibility
- **SQL92** Vendor extension.
- **Sybase** Not supported by Adaptive Server Enterprise.

See also
"HTML_ENCODE function [HTTP]" on page 309

"HTTP_ENCODE function [HTTP]" on page 310

HTML_ENCODE function [HTTP]

Function
Encodes special characters within strings to be inserted into HTML documents.

Syntax
HTML_ENCODE (*string*)

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
"	&quot;
'	&#39;
&	&amp;
<	&lt;
>	&gt;
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 309

"HTTP_ENCODE function [HTTP]" on page 310

HTTP_DECODE function [HTTP]

Function

Decodes special characters within strings for use with HTTP.

Syntax

HTTP_DECODE (*string*)

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 309

"HTTP_ENCODE function [HTTP]" on page 310

HTTP_ENCODE function [HTTP]

Function

Encodes special characters in strings for use with HTTP.

Syntax

HTML_ENCODE (*string*)

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

Character	Substitution
<	%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 309

"HTTP_DECODE function [HTTP]" on page 310

HTTP_HEADER function [HTTP]

Function

Gets the value of an HTTP header.

Syntax

HTML_HEADER (*field-name*)

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 311

"NEXT_HTTP_HEADER function [HTTP]" on page 331

"NEXT_HTTP_VARIABLE function [HTTP]" on page 332

HTTP_VARIABLE function [HTTP]

Function

Gets the value of an HTTP variable.

Syntax

HTML_VARIABLE (*var-name* [[, *instance*] , *header-field*)

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 309

"NEXT_HTTP_HEADER function [HTTP]" on page 331

"NEXT_HTTP_VARIABLE function [HTTP]" on page 332

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.

Examples

The following statement returns the indexed column name id, which is the first column of the multicolumn index with *index-id* equal to 6:

```
SELECT INDEX_COL( 'sales_order_items', 6, 0)
```

The following statement returns the indexed column name line_id, which is the second column of the multicolumn index with *index-id* equal to 6:

```
SELECT INDEX_COL( 'sales_order_items', 6, 1)
```

The following statement returns the indexed column name file_id, which is the only column in the single column index with *index-id* equal to 1:

```
SELECT INDEX_COL( 'ul_statement', 1, 0, 3)
```

The following statement returns the indexed column name quantity, which is the only column in the single column index with *index-id* equal to 4:

```
SELECT INDEX_COL( 'sales_order_items', 4, 0, 1)
```

Standards and compatibility

- **SQL92** Vendor extension.
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ.

See also

"OBJECT_ID function [System]" on page 337

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 datatype 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 datatype and size.

See "REPLACE function [String]" for more information.

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 373

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 Adaptive Server 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( 1000000000 ) 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]" on page 53

"ASE_FUNCTION_BEHAVIOR option" on page 44

"HEXTOINT function [Data type conversion]" on page 306

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 282

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* **)**

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 Adaptive Server 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.

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 datatype 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 datatype 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 323

"UCASE function [String]" on page 378

"UPPER function [String]" on page 379

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 datatype of a LEFTfunction 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 datatype 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 355

Chapter 11, “International Languages and Character Sets” in the *Sybase IQ System Administration Guide*

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.

BINARYand 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

Returns the characters:

```
select len(notes) from titles
where title_id = "PC9999"
-----------
39
```

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 4, "SQL Data Types."

Functions "CHAR_LENGTH function [String]" on page 280 and "STR_REPLACE function [String]" on page 370.

For general information about string functions, see "String functions" on page 264.

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 277

"CHAR_LENGTH function [String]" on page 280

Chapter 11, "International Languages and Character Sets" in the *Sybase IQ System Administration Guide*

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 322

Chapter 11, "International Languages and Character Sets" in the *Sybase IQ System Administration Guide*

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 to be searched for. 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.

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.

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 the string is not found, 0 is returned. Searching for a zero-length string returns 1. If any of the arguments are NULL, the result is NULL.

If multibyte characters are used, with the appropriate collation, then the starting position and the return value may be different from the *byte* positions.

Standards and compatibility

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

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 323

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 322

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 datatype 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 datatype 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 318

"UCASE function [String]" on page 378

"UPPER function [String]" on page 379

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 352 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 358

"TRIM function [String]" on page 376

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 employee table:

```
SELECT MAX ( salary )
FROM employee
```

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 325

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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 employee table:

```
SELECT MIN ( salary )
FROM employee
```

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 324

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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 278.

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 351

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" on page 65

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 278.

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. 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 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.

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 cannot use the NEWID function as a column default for 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 372

"UUIDTOSTR function [String]" on page 381

UNIQUEIDENTIFIER in "Binary data types" on page 229

UNIQUEIDENTIFIERSTR in Character data types on page 222

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* })

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* })

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 337

NEXT_HTTP_HEADER function [HTTP]

Function

Gets the next HTTP header name.

Syntax

NEXT_HTTP_HEADER (*header-name*)

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*)

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 309

“HTTP_VARIABLE function [HTTP]” on page 311

“NEXT_HTTP_HEADER function [HTTP]” on page 331

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 Adaptive Server Anywhere.

See also

“Analytical functions” on page 252

Chapter 4, “Using OLAP” in the *Sybase IQ Performance and Tuning Guide*

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 185

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 department
WHERE dept_id > 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 *customer* table:

```
SELECT OBJECT_ID ('CUSTOMER') 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 282

"DB_ID function [System]" on page 296

"OBJECT_NAME function [System]" on page 337

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 282

"DB_NAME function [System]" on page 296

"OBJECT_ID function [System]" on page 337

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 Adaptive Server Anywhere or Adaptive Server Enterprise.

See also
"BIT_LENGTH function [String]" on page 277

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 to be searched for. This string is limited to 255 bytes. 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 5-16 lists the pattern wildcards.

Table 5-16: 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

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 pattern is not found, it returns zero (0).

Standards and compatibility

- **SQL92** Vendor extension.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

"LIKE conditions" on page 193

"LOCATE function [String]" on page 321

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 Adaptive Server Anywhere.

See also

"Analytical functions" on page 252

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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 Adaptive Server Anywhere.

See also

“Analytical functions” on page 252

“NTILE function [Analytical]” on page 333

“PERCENTILE_DISC function [Analytical]” on page 343

Chapter 4, “Using OLAP” in the *Sybase IQ Performance and Tuning Guide*

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 Adaptive Server Anywhere.

See also

“Analytical functions” on page 252

“NTILE function [Analytical]” on page 333

“PERCENTILE_CONT function [Analytical]” on page 340

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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* })

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 269 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 269

PROPERTY_DESCRIPTION function [System]

Function

Returns a description of a property.

Syntax

PROPERTY_DESCRIPTION ({ *property-id* | *property-name* })

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 269, "Properties available for the server" on page 269, and "Properties available for each database" on page 270.

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 269

"Properties available for the server" on page 269

"Properties available for each database" on page 270

PROPERTY_NAME function [System]

Function
Returns the name of the property with the supplied property number.

Syntax
PROPERTY_NAME (*property-id*)

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 269

"Properties available for the server" on page 269

"Properties available for each database" on page 270

PROPERTY_NUMBER function [System]

Function
Returns the property number of the property with the supplied property name.

Syntax
PROPERTY_NUMBER (*property-name*)

Parameters
property-name A property name. For property names, see the lists in "Connection properties" on page 269, "Properties available for the server" on page 269, and "Properties available for each database" on page 270.

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 269

"Properties available for the server" on page 269

"Properties available for each database" on page 270

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 5-17 lists the dates in the quarters of the year.

Table 5-17: 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" on page 65

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 Adaptive Server Anywhere.

See also

“Analytical functions” on page 252

“DENSE_RANK function [Analytical]” on page 298

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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 327

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 datatype 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 datatype 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 353

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.SYSTABLE
WHERE table_id <= 5
```

Usage

The result datatype 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 datatype and size.

There are two ways to work around this issue:

- Declare a local temporary table and then do 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 373

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 datatype 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 datatype and size.

See "REPLACE function [String]" on page 352 for more information.

Standards and compatibility

- **SQL92** Vendor extension.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

"REPEAT function [String]" on page 351

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 323 and "UPPER function [String]" on page 379.

For general information about string functions, see “String functions” on page 264.

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 datatype 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 352 for more information.

Standards and compatibility

- **SQL92** Vendor extension.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also

“LEFT function [String]” on page 318

Chapter 11, “International Languages and Character Sets” in the *Sybase IQ System Administration Guide*

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 378

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( "PRODUCT" ) FROM PRODUCT
```

rowid(product)
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 PRODUCT.ID, ROWID ( PRODUCT )
FROM PRODUCT
WHERE PRODUCT.ID < 400
```

id	rowid(product)
300	1
301	2
302	3

The following statement deletes all rows with row ID values greater than 50:

```
DELETE FROM PRODUCT
WHERE ROWID ( PRODUCT ) > 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 352 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 324

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 278.

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 274

"COS function [Numeric]" on page 287

"COT function [Numeric]" on page 287

"TAN function [Numeric]" on page 376

SORTKEY function [String]

Function
Generates values that can be used to sort character strings based on alternate collation rules.

Syntax
SORTKEY (*string-expression* [, *collation-name*])

Parameters
string-expression The string expression may contain only characters that are encoded in the database's character set. String expression is usually one of the table fields upon which users want the query results to be ordered.

If *string-expression* is an empty string, SORTKEY returns a zero-length binary value. If *string-expression* is NULL, SORTKEY returns a null value. An empty string has a different sort order value than a null string from a database column.

The maximum length of the string that SORTKEY can handle is 254 bytes. Any longer part is ignored.

collation-name A string or character variable that specifies the name of the sort order to use.

If you do not specify a collation, the default is "thaidict". Although the following are all valid values for collation name, Sybase IQ is certified only with "thaidict".

Table 5-18: Collation name for sort order

Sort order description	Collation name
Binary sort	binary
Default Unicode multilingual	default
CP850 Alternative: no accent	altnoacc
CP850 Alternative: lowercase first	altdict
CP850 Alternative: no case preference	altnocsp
CP850 Scandinavian dictionary	scandict
CP850 Scandinavian: no case preference	scannocp
Latin-1 English, French, German dictionary	dict
Latin-1 English, French German no case	nocase
Latin-1 English, French German no case preference	nocasep
Latin-1 English, French German no accent	noaccent
Latin-1 Spanish dictionary	espdict
Latin-1 Spanish no case	espnocs
Latin-1 Spanish no accent	espnoac
Thai dictionary	thaidict

Examples

Assume the following table schema:

```
CREATE TABLE T1(id int, c1 varchar(40) shadowc1
varbinary(240))
```

SORTKEY() returns values in the sort order thaidict (Thai dictionary), the Thai character set in UTF8 form. The following statements generate the same result:

```
SELECT c1, SORTKEY(c1) from T1 where rid=3
SELECT c1, SORTKEY(c1, 'thaidict') from T1 where rid=3)
SELECT
'\340\270\201\340\271\207',SORTKEY('\340\279\201\340\2
71\207') from T1 where rid=3
```

Note Sybase IQ provides a utility to convert data files in CP874 format into UTF8 collation. For details, see "CP874toUTF8 utility" in Chapter 3, "Database Administration Utilities" in the *Sybase IQ Utility Guide*.

The following table shows SORTKEY results using thaidict:

c1 in ascii	c1 in binary	SORTKEY()
à, à1	\340\270\201\340\271\207	0x11a3011c

Usage

The following statements generate the same result. SORTKEY() returns values in the sort order default Unicode multilingual:

```
SELECT c1, SORTKEY(c1, 'dict') from T1 where rid=3
SELECT 'coop', SORTKEY('coop', 'dict') from T1 where
rid=3
```

The following table shows SORTKEY results using dict:

c1	SORTKEY()
0x08890997099709b30008000800080008	0x11a3011c

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 are beyond the limitation of collations supported by Sybase IQ. The returned value is a binary value that contains coded sort order information for the input string retained from the SORTKEY function.

For example, you can issue the following SELECT statement to retrieve 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 could instead store the value returned by SORTKEY in a column with the source character string. When you need to retrieve the character data in the desired order, the SELECT statement needs to include only an ORDER BY clause on the column that contains the results of running SORTKEY.

```
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.

The input of SORTKEY can generate up to six bytes of sort order information for each input character. The output of SORTKEY is of type varbinary and has a maximum length of (254 * 6) bytes.

Standards and compatibility

- **SQL92** Vendor extension.
- **Sybase** Compatible with Adaptive Server Enterprise, except that Adaptive Server Enterprise does not allow the use of self-defined sort orders.

See also

Chapter 11, "International Languages and Character Sets" in *Sybase IQ System Administration Guide*

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.

NULL Returns null values on one-element input sets in Sybase IQ versions prior to 12.7.

Examples — Given this data:

```
SELECT salary FROM employee WHERE dept_id = 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 employee
WHERE dept_id = 300
```

Given this data:

```
SELECT unit_price FROM product WHERE name = 'Tee Shirt'
```

name	unit_price
Tee Shirt	9.00
Tee Shirt	14.00
Tee Shirt	14.00

The following statement returns the value 2.88675134594813049:

```
SELECT STDDEV ( unit_price ) FROM product
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.

STDDEV does not support the keyword DISTINCT. A syntax error is returned if DISTINCT is used with STDDEV.

Standards and compatibility

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

See also

"STDDEV_SAMP function [Aggregate]" on page 368

"VARIANCE function [Aggregate]" on page 383

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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 sales_order_items 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 252

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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.

NULL Returns null values on one-element input sets in Sybase IQ versions prior to 12.7.

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 sales_order_items 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.

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 252

"STDDEV function [Aggregate]" on page 365

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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*.

```
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 4, "SQL Data Types."

Function "LENGTH function [String]" on page 320.

For general information about string functions, see "String functions" on page 264.

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.

This function is useful for inserting 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 329

"UUIDTOSTR function [String]" on page 381

UNIQUEIDENTIFIER in "Binary data types" on page 229

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 313

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 employee
```

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 275

"COUNT function [Aggregate]" on page 287

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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 375

"USER_ID function [System]" on page 379

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 375

"USER_NAME function [System]" on page 380

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 287

"SIN function [Numeric]" on page 361

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 352 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 324

“RTRIM function [String]” on page 358

TRUNCATE function [Numeric]

Function

Truncates a number at a specified number of places after the decimal point. Deprecated in favor of TRUNCNUM.

Syntax

"**TRUNCATE**" (*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 "TRUNCATE"( 655, -2 ) FROM iq_dummy
```

The following statement returns the value 655.340:

```
SELECT "TRUNCATE"( 655.348, 2 ) FROM iq_dummy
```

Usage

This function is the same as TRUNCNUM. Using TRUNCNUM is recommended, as it does not cause keyword conflicts.

The quotation marks are required because of a keyword conflict with the TRUNCATE TABLE statement. You can use TRUNCATE without the quotation marks only if the QUOTED_IDENTIFIER option is set to OFF.

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

“QUOTED_IDENTIFIER option [TSQL]” on page 143

"TRUNCNUM function [Numeric]" on page 378

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 355

"TRUNCATE function [Numeric]" on page 377

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 352 for more information.

Example — The following statement returns the value "CHOCOLATE":

```
SELECT UCASE( 'ChocoLate' ) FROM iq_dummy
```

Usage
The result datatype 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 datatype 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 318

"UPPER function [String]" on page 379

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 datatype 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 datatype and size.

Standards and compatibility
- **SQL92** This function is SQL92 compatible.
- **Sybase** Compatible with Adaptive Server Enterprise.

See also
"LCASE function [String]" on page 318

"LOWER function [String]" on page 323

"UCASE function [String]" on page 378

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 375

"USER_NAME function [System]" on page 380

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 375

"USER_ID function [System]" on page 379

UUIDTOSTR function [String]

Function
Converts a unique identifier value (UUID, also known as GUID) to a string value.

Syntax
NEWID*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:

```
SELECT UUIDTOSTR(uid_col),c1 FROM mytab
```

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 329

"STRTOUUID function [String]" on page 372

UNIQUEIDENTIFIER in "Binary data types" on page 229

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( ship_date ) AS Year, quarter( ship_date )
   AS Quarter, AVG( quantity ) AS Average,
   VAR_POP( quantity ) AS Variance
FROM sales_order_items 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 252

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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.

NULL Returns null values on one-element input sets in Sybase IQ versions prior to 12.7.

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,
   VAR_POP( quantity ) AS Variance
FROM sales_order_items 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.

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 252

"VARIANCE function [Aggregate]" on page 383

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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.

NULL Returns null values on one-element input sets in Sybase IQ versions prior to 12.7.

Examples

Given this data:

```
SELECT salary FROM employee WHERE dept_id = 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 employee
WHERE dept_id = 300
```

Given this data:

```
SELECT unit_price FROM product WHERE name = 'Tee Shirt'
```

name	unit_price
Tee Shirt	9.00
Tee Shirt	14.00
Tee Shirt	14.00

The following statement returns the value 8.33333333333334327:

```
SELECT VARIANCE ( unit_price ) FROM product
WHERE name = 'Tee Shirt'
```

Usage

The formula used to calculate VARIANCE is

$$var = \frac{n\sum x^2 - (\sum x)^2}{n(n-1)}$$

VARIANCE returns a result of data type double precision floating point. If applied to the empty set, the result is NULL.

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 365

"VAR_SAMP function [Aggregate]" on page 382

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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 278.

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.

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 customer_id, cust_last_name, credit_limit,
   WIDTH_BUCKET(credit_limit, 100, 5000, 10) "Credit
   Group"
   FROM customers WHERE territory = 'MA'
   ORDER BY "Credit Group";
```

```
CUSTOMER_ID  CUST_LAST_NAME  CREDIT_LIMIT  Credit Group
-----------  --------------  ------------  ------------
825          Dreyfuss                 500             1
826          Barkin                   500             1
853          Palin                    400             1
827          Siegel                   500             1
843          Oates                    700             2
844          Julius                   700             2
835          Eastwood                1200             3
840          Elliott                 1400             3
842          Stern                   1400             3
841          Boyer                   1400             3
837          Stanton                 1200             3
836          Berenger                1200             3
848          Olmos                   1800             4
849          Kaurusmdki              1800             4
828          Minnelli                2300             5
829          Hunter                  2300             5
852          Tanner                  2300             5
851          Brown                   2300             5
850          Finney                  2300             5
830          Dutt                    3500             7
831          Bel Geddes              3500             7
832          Spacek                  3500             7
838          Nicholson               3500             7
839          Johnson                 3500             7
833          Moranis                 3500             7
834          Idle                    3500             7
845          Fawcett                 5000            11
846          Brando                  5000            11
847          Streep                  5000            11
```

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 333, which creates equiheight histograms.

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

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 388

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 278.

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 timestamp 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 388

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 6

SQL Statements

About this chapter

This chapter presents an alphabetical listing of the SQL statements available in Sybase IQ, including some that can be used only from Embedded SQL or DBISQL.

Using the SQL statement reference

This section describes the conventions used in documenting the SQL statements.

Common elements in SQL syntax

This section lists language elements that are found in the syntax of many SQL statements.

For more information on the elements described here, see “Identifiers” on page 177; Chapter 4, “SQL Data Types,” “Search conditions” on page 189; “Expressions” on page 179; or “Strings” on page 178.

- column-name – an identifier that represents the name of a column.
- condition – an expression that evaluates to TRUE, FALSE, or UNKNOWN.
- connection-name – a string representing the name of an active connection.
- data-type – a storage data type.
- expression – an expression.
- filename – a string containing a file name.
- host-variable – a C language variable, declared as a host variable, preceded by a colon.

- indicator-variable – a second host variable of type short int immediately following a normal host variable. An indicator variable must also be preceded by a colon. Indicator variables are used to pass NULL values to and from the database.
- number – any sequence of digits followed by an optional decimal part and preceded by an optional negative sign. Optionally, the number can be followed by an 'e' and then an exponent. For example,

```
42
-4.038
.001
3.4e10
1e-10
```

- owner – an identifier representing the user ID who owns a database object.
- role-name – an identifier representing the role name of a foreign key.
- savepoint-name – an identifier that represents the name of a savepoint.
- search-condition – a condition that evaluates to TRUE, FALSE, or UNKNOWN.
- special-value – one of the special values described in "Special values" on page 205.
- statement-label – an identifier that represents the label of a loop or compound statement.
- table-list – a list of table names, which might include correlation names. For more information, see FROM clause on page 553.
- table-name – an identifier that represents the name of a table.
- userid – an identifier representing a user name. The user ID is not case sensitive and is unaffected by the setting of the CASE RESPECT property of the database.
- variable-name – an identifier that represents a variable name.

Syntax conventions

The following conventions are used in the SQL syntax descriptions:

- Keywords – All SQL keywords appear in UPPERCASE; however, SQL keywords are case insensitive, so you can type keywords in any case. For example, SELECT is the same as Select, which is the same as select.

- Placeholders – Items that must be replaced with appropriate identifiers or expressions are shown in *italics*.
- Continuation – Lines beginning with an ellipsis (...) are a continuation from the previous line.
- Optional portions – Optional portions of a statement are enclosed by square brackets. For example:

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

  This example indicates that the *savepoint-name* is optional. Do not type the square brackets.
- 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. When more than one is specified, they must be separated by commas if indicated as such. For example:

  ```
  UNIQUE (column-name [, ...])
  ```

  The example indicates that you can specify *column-name* more than once, separated by commas. Do not type the square brackets.
- Alternatives – When one option must be chosen, the alternatives are enclosed in curly braces. For example:

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

  The example indicates that if you choose the QUOTES option, you must provide one of ON or OFF. Do not type the braces.
- One or more options – If you choose more than one, separate your choices by commas. For example:

  { **CONNECT**, **DBA**, **RESOURCE** }

Statement applicability indicators

Some statement titles are followed by an indicator in square brackets that shows where the statement can be used. These indicators are as follows:

- [ESQL] – The statement is for use in Embedded SQL.
- [DBISQL] – The statement is for use only in DBISQL.
- [SP] – The statement is for use in stored procedures or batches.

- [TSQL] – The statement is implemented for compatibility with Adaptive Server Enterprise. In some cases, the statement cannot be used in stored procedures that are not Transact-SQL format. In other cases, there is an alternative statement that is closer to the SQL92 standard that is recommended unless Transact-SQL compatibility is an issue.

If two sets of brackets are used, the statement can be used in both environments. For example, [ESQL] [SP] means a statement can be used either in Embedded SQL or in stored procedures.

ALLOCATE DESCRIPTOR statement [ESQL]

Description

Allocates space for a SQL descriptor area (SQLDA).

Syntax

ALLOCATE DESCRIPTOR *descriptor-name*
... [WITH MAX { *integer* | *host-variable* }]

Parameters

descriptor-name:
string

For more information, see Chapter 3, “SQL Language Elements.”

Examples

The following sample program includes an example of ALLOCATE DESCRIPTOR statement usage.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


EXEC SQL INCLUDE SQLCA;


#include <sqldef.h>


EXEC SQL BEGIN DECLARE SECTION;
int       x;
short     type;
int       numcols;
char      string[100];
a_sql_statement_number  stmt = 0;
EXEC SQL END DECLARE SECTION;

int main(int argc, char * argv[])
```

```
{
    struct sqlda *      sqlda1;

    if( !db_init( &sqlca ) ) {
       return 1;
    }
    db_string_connect(&sqlca,
"UID=dba;PWD=sql;DBF=d:\\ASIQ-12_5\\sample.db");

    EXEC SQL ALLOCATE DESCRIPTOR sqlda1 WITH MAX 25;

    EXEC SQL PREPARE :stmt FROM 'select * from
employee';
    EXEC SQL DECLARE curs CURSOR FOR :stmt;
    EXEC SQL OPEN curs;

    EXEC SQL DESCRIBE :stmt into sqlda1;
    EXEC SQL GET DESCRIPTOR sqlda1 :numcols=COUNT;
       // how many columns?
    if( numcols > 25 ) {
       // reallocate if necessary
       EXEC SQL DEALLOCATE DESCRIPTOR sqlda1;
       EXEC SQL ALLOCATE DESCRIPTOR sqlda1
          WITH MAX :numcols;
    }
    type = DT_STRING;   // change the type to string
    EXEC SQL SET DESCRIPTOR sqlda1 VALUE 2 TYPE = :type;
    fill_sqlda( sqlda1 );  // allocate space for the
variables

    EXEC SQL FETCH ABSOLUTE 1 curs USING DESCRIPTOR
sqlda1;
    EXEC SQL GET DESCRIPTOR sqlda1 VALUE 2 :string =
DATA;

    printf("name = %s", string );

    EXEC SQL DEALLOCATE DESCRIPTOR sqlda1;
    EXEC SQL CLOSE curs;
    EXEC SQL DROP STATEMENT :stmt;

    db_string_disconnect( &sqlca, "" );
    db_fini( &sqlca );

    return 0;
}
```

Usage

You must declare the following in your C code prior to using this statement:

```
struct sqlda * descriptor_name
```

The WITH MAX clause lets you specify the number of variables within the descriptor area. The default size is 1.

You must still call fill_sqlda to allocate space for the actual data items before doing a fetch or any statement that accesses the data within a descriptor area.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

See also

DEALLOCATE DESCRIPTOR statement [ESQL] on page 514

The SQL descriptor area (SQLDA) in the *Adaptive Server Anywhere Programming Guide*

ALTER DATABASE statement

Description

Upgrades a database created with a previous version of the software or adds or removes Java or jConnect support. Run this statement with Interactive SQL Java.

Syntax

ALTER DATABASE
UPGRADE
[**JAVA** { **ON** | **OFF**
| **JDK** { **‘ 1.1.8 ‘** | **‘ 1.3 ’** } }]
[**JCONNECT** { **ON** | **OFF**}]
| **REMOVE JAVA**

Examples

Upgrade a database created with the Java options off:

```
ALTER DATABASE UPGRADE JAVA OFF JCONNECT OFF
```

Usage

The ALTER DATABASE statement upgrades databases created with earlier versions of the software. This applies to maintenance releases as well as major releases. For example, you can upgrade a database created with version 12.6 to 12.7.

Note See the *Sybase IQ Installation and Configuration Guide* for backup recommendations before you upgrade.

When you upgrade a database, Sybase IQ makes the following changes:

- Upgrades the system tables to the current version.
- Adds any new database options.

You can also use ALTER DATABASE UPGRADE simply to add Java or jConnect features if the database was created with the current version of the software.

Warning! Be sure to start the server in a way that restricts user connections before you run ALTER DATABASE UPGRADE. For instructions and other upgrade caveats, see the chapter "Migrating Data," in the *Sybase IQ Installation and Configuration Guide* for your platform.

After using ALTER DATABASE UPGRADE, shut down the database.

Note If upgrade of a Sybase IQ 12.6 database returns a "Database upgrade not possible" error, see "Insufficient procedure identifiers," in *Sybase IQ Troubleshooting and Recovery Guide*.

JAVA clause Controls support for Java in the upgraded database.

- Specify JAVA ON to enable support for Java in the database by adding entries for the default Sybase runtime Java classes to the system tables. If Java in the database is already installed, but is at a lower version than the default classes, this clause upgrades it to the current default classes. The default classes are the JDK 1.3 classes.
- Specify JAVA OFF to prevent the addition of Java in the database to databases that do not already have it installed. For databases that already have Java installed, setting JAVA OFF does not remove Java support: the version of Java remains at the current version. To remove Java from the database, use the REMOVE JAVA clause.
- Specify JAVA JDK '1.1.8' or JAVA JDK '1.3' to install support for the named version of the JDK.

  The ALTER DATABASE UPGRADE statement only upgrades your database to a higher version of JDK. To downgrade, first remove Java from the database, then add it back with the lower JDK version. For example, to downgrade from JDK 1.3 to JDK 1.1.8:

  ```
  ALTER DATABASE REMOVE JAVA
  ALTER DATABASE UPGRADE JAVA JDK '1.1.8'
  ```

  Classes for JDK 1.1.8 are stored in *java/1.1/classes.zip* under the Sybase IQ installation directory. Classes for JDK 1.3 are stored in *java/1.3/rt.jar*.

The default behavior is JAVA OFF.

To use Java after adding it in the database, you must restart the database.

JCONNECT clause To allow the Sybase jConnect JDBC driver to access system catalog information, you must specify JCONNECT ON. This installs jConnect system tables and procedures. To exclude the jConnect system objects, specify JCONNECT OFF. You can still use JDBC, as long as you do not access system catalog information. The default is to include jConnect support (JCONNECT ON).

REMOVE JAVA clause Removes Java from a database. The operation leaves the database as if it were created with JAVA OFF. When the statement is issued Java in the database must not be in use. Remove all Java classes from the database before executing this statement. The statement ignores stored procedures and triggers that reference Java objects, and the presence of these objects does not trigger an error in the ALTER DATABASE statement.

Side effects

- Automatic commit

Standards

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

Permissions

Must have DBA authority.

See also

CREATE DATABASE statement on page 442

"Migrating Data" in the *Sybase IQ Installation and Configuration Guide*

"Introduction to Java in the Database" in the *Adaptive Server Anywhere Programming Guide*

ALTER DBSPACE statement

Description

Changes the read/write mode, changes the size, or extends an existing dbspace.

Syntax

ALTER DBSPACE *dbspace-name*
{ **READWRITE** | **READONLY** | **RELOCATE**
| **SIZE** *dbspace-size* [KB | MB | GB | TB | PAGES]
| **ADD** *dbspace-size* [KB | MB | GB | TB | PAGES] }

Examples

Example 1 Change the mode of a dbspace called *mydb_tmp_2* to relocate.

```
ALTER DBSPACE mydb_tmp_2 RELOCATE;
```

Example 2 Specify the new size of 10MB for the dbspace IQ_SYSTEM_MAIN.

```
ALTER DBSPACE IQ_SYSTEM_MAIN SIZE 10MB
```

Example 3 Increase the size of the dbspace IQ_SYSTEM_TEMP by 2GB.

```
ALTER DBSPACE IQ_SYSTEM_TEMP ADD 2 GB
```

Example 4 Specify the new size of 4MB for the dbspace IQ_SYSTEM_TEMP. (SIZE defaults to megabytes.)

```
ALTER DBSPACE IQ_SYSTEM_TEMP SIZE 4
```

Example 5 Increase the size of the dbspace IQ_SYSTEM_MAIN by 1000 pages. (ADD defaults to pages.)

```
ALTER DBSPACE IQ_SYSTEM_MAIN ADD 1000
```

Usage

The ALTER DBSPACE statement changes the read/write mode, changes the size, or extends an existing dbspace. The sp_iqdbspace system stored procedure displays the mode and size of the dbspace. Dbspace names are case sensitive for databases created with CASE RESPECT.

READWRITE clause Specifies that allocations can be made from the dbspace. The mode of a newly created dbspace is readwrite.

READONLY clause Specifies that the server no longer writes to the dbspace. You can still modify objects on the dbspace, but new versions are placed on the remaining readwrite dbspaces.

RELOCATE clause Specifies that space is not allocated from the dbspace and that the objects on the dbspace are subject to relocation. The server does not write to an IQ Main dbspace in relocate mode.

SIZE clause Specifies the new size of the dbspace in units of pages, kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). The default is megabytes. You can increase the size of the dbspace only if the free list (an allocation map) has sufficient room or if the dbspace has sufficient reserved space. You can decrease the size of the dbspace only if the truncated portion is not in use.

ADD clause ALTER DBSPACE ADD extends the dbspace by the specified *dbspace-size* in units of pages, kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). The default is PAGES. The page size of a database is fixed when the database is created.

Note You can increase dbspace size only if the dbspace has sufficient reserved space.

You can also view and change the dbspace mode and size through the Sybase Central Dbspaces window.

Side effects

- Automatic commit
- Automatic checkpoint
- A mode change to READONLY or RELOCATE causes immediate relocation of the internal database structures on the dbspace to one of the read/write dbspaces.

Standards

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

Permissions

Must have DBA authority.

See also

CREATE DBSPACE statement on page 453

CREATE DATABASE statement on page 442

DROP statement on page 533

"sp_iqdbspace procedure" in Chapter 10, "System Procedures"

"Working with dbspaces" in Chapter 5, "Working with Database Objects," of the *Sybase IQ System Administration Guide*

ALTER DOMAIN statement

Description

Renames a user-defined domain or data type. Does not rename Java types.

Syntax

ALTER { **DOMAIN** | **DATATYPE** } *user-type*
RENAME *new-name*

Parameters

new-name:
an identifier representing the new domain name.

user-type:
user-defined data type of the domain being renamed.

Examples

The following renames the Address domain to MailingAddress:

```
ALTER DOMAIN Address RENAME MailingAddress
```

Usage

The ALTER DOMAIN statement updates the name of the user-defined domain or data type in the SYSUSERTYPE system table.

You must recreate any procedures, views or events that reference the user-defined domain or data type, or else they will continue to reference the former name.

Side effects

Automatic commit.

Permissions

Must have DBA authority or be the database user who created the domain.

See also

CREATE DOMAIN statement on page 456

Chapter 4, “SQL Data Types”

“SYSUSERTYPE system table” on page 734

ALTER EVENT statement

Description

Changes the definition of an event or its associated handler for automating predefined actions. Also alters the definition of scheduled actions.

Syntax

ALTER EVENT *event-name*
[**DELETE TYPE** | **TYPE** *event-type*]
{ **WHERE** { *trigger-condition* | **NULL** }
| { **ADD** | [**MODIFY**] | **DELETE** } **SCHEDULE** *schedule-spec*
}
[**ENABLE** | **DISABLE**]
[[**MODIFY**] **HANDLER** *compound-statement* | **DELETE HANDLER** }

Parameters

event-type:

BackupEnd | "Connect"
| ConnectFailed | DatabaseStart
| DBDiskSpace | "Disconnect"
| GlobalAutoincrement | GrowDB
| GrowLog | GrowTemp
| LogDiskSpace | "RAISERROR"
| ServerIdle | TempDiskSpace

trigger-condition:

[event_condition(*condition-name*) { = | < | > | != | <= | >= } *value*]

schedule-spec:
[*schedule-name*]
{ START TIME *start-time* | BETWEEN *start-time* AND *end-time* }
[EVERY *period* { HOURS | MINUTES | SECONDS }]
[ON { (*day-of-week, ...*) | (*day-of-month, ...*) }]
[START DATE *start-date*]

event-name | *schedule-name*:
identifier

day-of-week :
string

value | *period* | *day-of-month* :
integer

start-time | *end-time* :
time

start-date :
date

Usage

The ALTER EVENT statement lets you alter an event definition created with CREATE EVENT. Possible uses include the following:

- Use ALTER EVENT to change an event handler during development.
- Define and test an event handler without a trigger condition or schedule during a development phase, and then add the conditions for execution using ALTER EVENT once the event handler is completed.
- Disable an event handler temporarily by disabling the event.

When you alter an event using ALTER EVENT, specify the event name and, optionally, the schedule name.

List event names by querying the system table SYSEVENT. For example:

```
SELECT event_id, event_name FROM SYS.SYSEVENT
```

List schedule names by querying the system table SYSSCHEDULE. For example:

```
SELECT event_id, sched_name FROM SYS.SYSSCHEDULE
```

Each event has a unique event ID. Use the event_id columns of SYSEVENT and SYSSCHEDULE to match the event to the associated schedule.

DELETE TYPE clause Removes an association of the event with an event type.

ADD | MODIFY | DELETE SCHEDULE clause Changes the definition of a schedule. Only one schedule can be altered in any one ALTER EVENT statement.

WHERE clause The WHERE NULL option deletes a condition.

For descriptions of most of the parameters, see the CREATE EVENT statement on page 458.

Side effects

Automatic commit.

Permissions

Must have DBA authority.

See also

BEGIN... END statement on page 422

CREATE EVENT statement on page 458

Chapter 18, "Automating Tasks Using Schedules and Events," in the *Sybase IQ System Administration Guide*

ALTER INDEX statement

Description

Renames indexes in base or global temporary tables and foreign key role names of indexes and foreign keys explicitly created by a user.

Syntax

Syntax 1

ALTER INDEX *index-name rename-spec*

Syntax 2

ALTER [INDEX] FOREIGN KEY *role-name rename-spec*

Parameters

rename-spec:
ON [*owner*.]*table-name* RENAME [AS | TO] *new-name*

Examples

Example 1 Renames an index COL1_HG_OLD in the table jak.mytable to COL1_HG_NEW:

```
ALTER INDEX COL1_HG_OLD ON jak.mytable
RENAME AS COL1_HG_NEW
```

Example 2 Renames a foreign key role name ky_dept_id in table dba.employee to emp_dept_id:

```
ALTER INDEX FOREIGN KEY ky_dept_id
ON dba.employee
RENAME TO emp_dept_id
```

Usage

The ALTER INDEX statement renames indexes and foreign key role names of indexes and foreign keys that were explicitly created by a user. Only indexes on base tables or global temporary tables can be renamed. You cannot rename indexes created to enforce key constraints.

ON claus The ON clause specifies the name of the table that contains the index or foreign key to rename.

RENAME [AS | TO] claus The RENAME clause specifies the new name of the index or foreign key role.

Note Attempts to alter an index in a local temporary table return the error "index not found." Attempts to alter a nonuser-created index, such as a default index (FP), return the error "Cannot alter index. Only indexes in base tables or global temporary tables with an owner type of USER can be altered."

Side Effects

Automatic commit. Clears the Results tab in the Results pane in Interactive SQL. Closes all cursors for the current connection.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Not supported by Adaptive Server Enterprise.

Permissions

Must own the table, or have REFERENCES permissions on the table, or have DBA authority.

See also

ALTER TABLE statement on page 409

CREATE INDEX statement on page 473

CREATE TABLE statement on page 499

ALTER PROCEDURE statement

Description

Replaces a procedure with a modified version. You must include the entire new procedure in the ALTER PROCEDURE statement, and reassign user permissions on the procedure.

Syntax

ALTER PROCEDURE [*owner*.]*procedure-name procedure-definition*

Parameters

procedure-definition:
CREATE PROCEDURE syntax following the name

Usage

The ALTER PROCEDURE statement is identical in syntax to the CREATE PROCEDURE statement.

Existing permissions on the procedure are maintained and need not be reassigned. If a DROP procedure and CREATE PROCEDURE were carried out, execute permissions would have to be reassigned.

Side effects

Automatic commit is a side effect of this statement.

Standards

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

Permissions

Must be the owner of the procedure or a DBA. Automatic commit.

See also

CREATE PROCEDURE statement on page 485

ALTER SERVER statement

Description

Modifies the attributes of a remote server.

Syntax

ALTER SERVER *server-name*
[**CLASS** '*server-class*']
[**USING** '*connection-info*']
[**CAPABILITY** '*cap-name*' { **ON** | **OFF** }]

Parameters

server-class:
{*ASAJDBC* | *ASEJDBC*
| *ASAODBC* | *ASEODBC*
| *DB2ODBC* | *MSSODBC*
| *ORAODBC* | *ODBC* }

connection-info:
{ *machine-name:port-number* [/*dbname*] | *data-source-name* }

cap-name:
the name of a server capability

Examples

- Changes the server class of the Adaptive Server named ase_prod so its connection to Sybase IQ is ODBC-based. The Data Source Name is ase_prod.

```
ALTER SERVER ase_prod
CLASS 'ASEODBC'
USING 'ase_prod'
```

- Changes a capability of server infodc:

```
ALTER SERVER infodc
CAPABILITY 'insert select' OFF
```

Usage

Changes made by ALTER SERVER do not take effect until the next connection to the remote server.

CLASS clause Use the CLASS clause to change the server class. For more information on server classes, see Chapter 17, “Server Classes for Remote Data Access” and Chapter 16, “Accessing Remote Data” in the *Sybase IQ System Administration Guide*.

USING clause The USING clause changes the server’s connection information. For more information about connection information, see CREATE SERVER statement on page 494.

CAPABILITY clause The CAPABILITY clause turns a server capability ON or OFF. Server capabilities are stored in the system table SYSCAPABILITY. The names of these capabilities are stored in the system table SYSCAPABILITYNAME. The SYSCAPABILITY table contains no entries for a remote server until the first connection is made to that server. At the first connection, Sybase IQ interrogates the server about its capabilities and then populates SYSCAPABILITY. For subsequent connections, the server’s capabilities are obtained from this table.

In general, you need not alter a server’s capabilities. It might be necessary to alter capabilities of a generic server of class ODBC.

Side Effects

Automatic commit is a side effect of this statement.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions

Must have RESOURCE authority.

See also

CREATE SERVER statement on page 494

DROP SERVER statement on page 538

Chapter 17, “Server Classes for Remote Data Access,” and Chapter 16, “Accessing Remote Data,” in the *Sybase IQ System Administration Guide*

ALTER SERVICE statement

Description

Alters a Web service.

Syntax

ALTER SERVICE *service-name*
[**TYPE** '*service-type-string*']
[*attributes*]
[**AS** *statement*']

Parameters

attributes:[AUTHORIZATION { ON | OFF }] [SECURE { ON | OFF }] [USER *user-name* | NULL }] [URL [PATH] { PATH } {ON | OFF | ELEMENTS }] [USING (*SOAP-prefix* | NULL }]

service-type-string: {'RAW' | 'HTML' | 'XML' | 'SOAP' | 'DISH'}

Examples

To set up a Web server quickly, start a database server with the -xs switch, then execute the following statements:

```
CREATE SERVICE tables TYPE 'HTML'

ALTER SERVICE tables
AUTHORIZATION OFF
USER DBA
AS SELECT * FROM SYS.SYSTABLE
```

After executing these statements, use any Web browser to open the URL http://localhost/tables.

Usage

The alter service statement causes the database server to act as a Web server.

service-name You cannot rename Web services.

service-type-string Identifies the type of the service. The type must be one of the listed service types. There is no default value.

AUTHORIZATION clause Determines whether users must specify a user name and password when connecting to the service. If authorization is OFF, the AS clause is required and a single user must be identified by the USER clause. All requests are run using that user's account and permissions.

If authorization is ON, all users must provide a user name and password. Optionally, you might limit the users that are permitted to use the service by providing a user or group name using the USER clause. If the user name is NULL, all known users can access the service.

The default value is ON. It is recommended that production systems be run with authorization turned on and that you grant permission to use the service by adding users to a group.

SECURE clause Indicates whether unsecure connections are accepted. ON indicates that only HTTPS connections are to be accepted. Service requests received on the HTTP port are automatically redirected to the HTTPS port. If set to OFF, both HTTP and HTTPS connections are accepted. The default value is OFF.

USER clause If authorization is disabled, this parameter becomes mandatory and specifies the user id used to execute all service requests. If authorization is enabled (the default), this optional clause identified the user or group permitted access to the service. The default value is NULL, which grants access to all users.

URL clause Determines whether URI paths are accepted and, if so, how they are processed. OFF indicates that nothing must follow the service name in a URI request. ON indicates that the remainder of the URI is interpreted as the value of a variable named url. ELEMENTS indicates that the remainder of the URI path is to be split at the slash characters into a list of up to 10 elements. The values are assigned to variables named url plus a numeric suffix of between 1 and 10; for example, the first three variable names are url1, url2, and url3. If fewer than 10 values are supplied, the remaining variables are set to NULL. If the service name ends with the character /, then URL must be set to OFF. The default value is OFF.

USING clause This clause applies only to DISH services. The parameter specifies a name prefix. Only SOAP services whose names begin with this prefix are handled.

statement If the statement is NULL, the URI must specify the statement to be executed. Otherwise, the specified SQL statement is the only one that can be executed through the service. SOAP services must have statements; DISH services must have none. The default value is NULL.

It is strongly recommended that all services run in production systems define a statement. The statement can be NULL only if authorization is enabled.

RAW The result set is sent to the client without any further formatting. You can produce formatted documents by generating the required tags explicitly within your procedure.

HTML The result set of a statement or procedure is automatically formatted into an HTML document that contains a table.

XML The result set is assumed to be in XML format. If it is not already so, it is automatically converted to XML RAW format.

SOAP The request must be a valid Simple Object Access Protocol, or SOAP, request. The result set is automatically formatted as a SOAP response. For more information about the SOAP standards, see www.w3.org/TR/SOAP at http://www.w3.org/TR/SOAP.

DISH A Determine SOAP Handler, or DISH, service acts as a proxy for one or more SOAP services. In use, it acts as a container that holds and provides access to a number of SOAP services. A Web Services Description Language (WSDL) file is automatically generated for each of the included SOAP services. The included SOAP services are identified by a common prefix, which must be specified in the USING clause.

Standards

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

Permissions

Must have DBA authority.

See also

CREATE SERVICE statement on page 496

DROP SERVICE statement on page 539

"Using the Built-in Web Server" in the *Adaptive Server Anywhere Database Administration Guide*

ALTER TABLE statement

Description

Modifies a table definition.

Syntax

ALTER TABLE [*owner*.]*table-name*
{ *add-clause* | *modify-clause* | *drop-clause* | *rename-clause* }

Parameters

add-clause:
ADD *column-definition* [*column-constraint*]...
| ADD *table-constraint*

modify-clause:
MODIFY *column-definition*
| MODIFY *column-name* [IDENTITY | DEFAULT *default-value*]
[NOT] NULL
| MODIFY *column-name* [CONSTRAINT *constraint-name*] CHECK NULL
| MODIFY *column-name* CHECK (*new-condition*)
ALTER *column-name column-modification*
| ALTER CONSTRAINT *constraint-name* CHECK (*new-condition*)

drop-clause:
{ DELETE | DROP } *column-name*
| { DELETE | DROP } CHECK
| { DELETE | DROP } CONSTRAINT *constraint-name*
| { DELETE | DROP } UNIQUE (*column-name* [, ...])
| { DELETE | DROP } PRIMARY KEY
| { DELETE | DROP } FOREIGN KEY *role-name*

rename-clause:
RENAME *new-table-name*
| RENAME *column-name* TO *new-column-name*
| RENAME *constraint-name* TO *new-constraint-name*

column-definition:
column-name data-type [NOT NULL]
[DEFAULT *default-value* | IDENTITY]

column-constraint:
[CONSTRAINT *constraint-name*] { UNIQUE
| PRIMARY KEY
| REFERENCES *table-name* [(*column-name*)] [*actions*]
| CHECK (*condition*)
| IQ UNIQUE (*integer*) }

default-value:
special-value
| *string*
| *global variable*
| [-] *number*
| (*constant-expression*)
| *built-in-function* (*constant-expression*)
| AUTOINCREMENT
| NULL
| TIMESTAMP
| LAST USER
| USER

special-value:
CURRENT { DATABASE | DATE | REMOTE USER | TIME
| TIMESTAMP | USER | PUBLISHER }

table-constraint:
{ UNIQUE (*column-name* [, ...])
| PRIMARY KEY (*column-name* [, ...])
| *foreign-key-constraint*
| CHECK (*condition*)}

foreign-key-constraint:
FOREIGN KEY [*role-name*] [(*column-name* [, ...])]
... REFERENCES *table-name* [(*column-name* [, ...])]
... [*actions*] [

actions:
[ON {UPDATE | DELETE} *action*]

action:
{ RESTRICT }

Examples

- Adds a new column to the *employees* table showing which office they work in:

```
ALTER TABLE employee
ADD office CHAR(20)
```

- Drops the *office* column from the *employees* table:

```
ALTER TABLE employee
DELETE office
```

- Adds a column to the *customer* table assigning each customer a sales contact:

```
ALTER TABLE customer
ADD sales_contact INTEGER
REFERENCES employee (emp_id)
```

- Adds a new column *cust_num* to the *customer* table and assigns a default value of 88:

```
ALTER TABLE customer
ADD cust_num INTEGER DEFAULT 88
```

Usage

The ALTER TABLE statement changes table attributes (column definitions and constraints) in a table that was previously created. The syntax allows a list of alter clauses; however, only one table constraint or column constraint can be added, modified, or deleted in each ALTER TABLE statement.

Note You cannot alter local temporary tables, but you can alter global temporary tables when they are in use by only one connection.

Sybase IQ enforces REFERENCES and CHECK constraints. Table and/or column check constraints added in an ALTER TABLE statement are not evaluated as part of that alter table operation. For details about CHECK constraints, see CREATE TABLE statement on page 499.

If SELECT * is used in a view definition and you alter a table referenced by the SELECT *, then you must run ALTER VIEW <viewname> RECOMPILE to ensure that the view definition is correct and to prevent unexpected results when querying the view.

ADD column-definition [column-constraint] Add a new column to the table. The table must be empty to specify NOT NULL. The table might contain data when you add an IDENTITY or DEFAULT AUTOINCREMENT column. If the column has a default IDENTITY value, all rows of the new column are populated with sequential values. You can also add a foreign key constraint as a column constraint for a single column key. The value of the IDENTITY/DEFAULT AUTOINCREMENT column uniquely identifies every row in a table. The IDENTITY/DEFAULT AUTOINCREMENT column stores sequential numbers that are automatically generated during inserts and updates. DEFAULT AUTOINCREMENT columns are also known as IDENTITY columns. When using IDENTITY/DEFAULT AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type, with scale 0. See CREATE TABLE statement on page 499 for more about column constraints and IDENTITY/DEFAULT AUTOINCREMENT columns.

Note You cannot add foreign key constraints to an unenforced primary key created with Sybase IQ version 12.4.3 or earlier.

ADD table-constraint Add a constraint to the table. You can also add a foreign key constraint as a table constraint for a single-column or multicolumn key. See CREATE TABLE statement on page 499 for a full explanation of table constraints.

If PRIMARY KEY is specified, the table must not already have a primary key created by the CREATE TABLE statement or another ALTER TABLE statement.

Note You cannot MODIFY a table or column constraint. To change a constraint, DELETE the old constraint and ADD the new constraint.

MODIFY column-name [NOT] NULL Change the NOT NULL constraint on the column to allow or disallow NULL values in the column.

MODIFY column-name [DEFAULT default-value | IDENTITY] The value of the IDENTITY or DEFAULT AUTOINCREMENT column uniquely identifies every row in a table. The IDENTITY/DEFAULT AUTOINCREMENT column stores sequential numbers that are automatically generated during inserts and updates. DEFAULT AUTOINCREMENT columns are also known as IDENTITY columns. When you use IDENTITY/DEFAULT AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type, with scale 0. See CREATE TABLE statement on page 499 for a full explanation of column constraints and IDENTITY/DEFAULT AUTOINCREMENT columns.

ALTER TABLE also supports the modification of column default values other than IDENTITY/DEFAULT AUTOINCREMENT. When modifying a column of a table, you can specify a default value for the column using the DEFAULT keyword. If a DEFAULT value is specified for a column, this DEFAULT value is used as the value of the column in any INSERT (or LOAD) statement that does not specify a value for the column.

For detailed information on the use of column DEFAULT values, see "Using column defaults" in Chapter 9, "Ensuring Data Integrity" in the *Sybase IQ System Administration Guide*.

MODIFY column-name CHECK NULL Delete the check constraint for the column.

MODIFY column-name CHECK (new-condition) Replace the existing CHECK condition for the column with the one specified.

ALTER column-name column-modification Change the definition of a column. The permitted modifications are as follows:

- **SET DEFAULT default-value** Change the default value of an existing column in a table. You can also use the MODIFY clause for this task, but ALTER is SQL92 compliant, and MODIFY is not. Modifying a default value does not change any existing values in the table.
- **DROP DEFAULT** Remove the default value of an existing column in a table. You can also use the MODIFY clause for this task, but ALTER is SQL92 compliant, and MODIFY is not. Dropping a default does not change any existing values in the table.
- **ADD** Add a named constraint or a CHECK condition to the column. The new constraint or condition applies only to operations on the table after its definition. The existing values in the table are not validated to confirm that they satisfy the new constraint or condition.

- **CONSTRAINT column-constraint-name** The optional column constraint name lets you modify or drop individual constraints at a later time, rather than having to modify the entire column constraint.
- **SET COMPUTE (expression)** Change the expression associated with a computed column. The values in the column are recalculated when the statement is executed, and the statement fails if the new expression is invalid.
- **DROP COMPUTE** Change a column from being a computed column to being a noncomputed column. This statement does not change any existing values in the table.

DELETE column-name Delete the column from the table. If the column is contained in any multicolumn index, uniqueness constraint, foreign key, or primary key, then the index, constraint, or key must be deleted before the column can be deleted. This does not delete CHECK constraints that refer to the column. An IDENTITY/DEFAULT AUTOINCREMENT column can only be deleted if IDENTITY_INSERT is turned off and the table is not a local temporary table.

DELETE CHECK Delete all check constraints for the table. This includes both table check constraints and column check constraints.

DELETE UNIQUE (column-name,...) Delete a uniqueness constraint for this table. Any foreign keys referencing this uniqueness constraint (rather than the primary key) are also deleted. Reports an error if there are associated foreign-key constraints. Use ALTER TABLE to delete all foreign keys that reference the primary key before you delete the primary key constraint.

DELETE PRIMARY KEY Delete the primary key constraint for this table. All foreign keys referencing the primary key for this table are also deleted. Reports an error if there are associated foreign key constraints. If the primary key is unenforced, DELETE returns an error if associated unenforced foreign key constraints exist.

DELETE FOREIGN KEY role-name Delete the foreign key constraint for this table with the given role name. Retains the implicitly created nonunique HG index for the foreign key constraint. Users can explicitly remove the HG index with the DROP INDEX statement.

RENAME new-table-name Change the name of the table to the *new-table-name*. Any applications using the old table name must be modified. Also, any foreign keys that were automatically assigned the same name as the old table name do not change names.

RENAME column-name TO new-column-name Change the name of the column to the *new-column-name*. Any applications using the old column name must be modified.

RENAME constraint-name TO new-constraint-name Change the name of the constraint to the *new-constraint-name*. Any applications using the old constraint name must be modified.

ALTER TABLE is prevented whenever the statement affects a table that is currently being used by another connection. ALTER TABLE can be time consuming, and the server does not process requests referencing the same table while the statement is being processed.

Side effects

- Automatic commit. The MODIFY and DELETE options close all cursors for the current connection. The DBISQL data window is also cleared.
- A checkpoint is carried out at the beginning of the ALTER TABLE operation.
- Once you alter a column or table, any stored procedures, views or other items that refer to the altered column no longer work.

Standards

- **SQL92** Intermediate-level feature. MODIFY clauses are not SQL92 compliant.
- **Sybase** Some clauses are supported by Adaptive Server Enterprise.

Permissions

Must be the owner of the table or have DBA authority or ALTER permission on the table. Requires exclusive access to the table.

See also

CREATE TABLE statement on page 499

DROP statement on page 533

“IDENTITY_INSERT option” on page 85

Chapter 4, “SQL Data Types”

ALTER VIEW statement

Description

Replaces a view definition with a modified version. You must include the entire new view definition in the ALTER VIEW statement.

Syntax

ALTER VIEW
... [*owner*.]*view-name* [(*column-name* [, ...])]
... **AS** *select-without-order-by*
... [**WITH CHECK OPTION**]

Usage

The ALTER VIEW statement is identical in syntax to the CREATE VIEW statement. ALTER VIEW replaces the entire contents of CREATE VIEW with the contents of ALTER VIEW. Existing permissions on the view are maintained, and need not be reassigned. If a DROP VIEW followed by CREATE VIEW is used, instead of ALTER VIEW, permissions on the view must be reassigned.

Note If SELECT * is used in a view definition and a table referenced by the SELECT * is altered, then you must run ALTER VIEW <*viewname*> RECOMPILE to ensure that the view definition is correct and to prevent unexpected results when the view is queried.

Side Effects

Automatic commit.

Standards

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

Permissions

Must be owner of the view or have DBA authority.

See also

CREATE VIEW statement on page 512

DROP statement on page 533

BACKUP statement

Description

Backs up a Sybase IQ database on one or more archive devices.

Syntax

BACKUP DATABASE
... [**CRC** { **ON** | **OFF** }]
... [**ATTENDED** { **ON** | **OFF** }]
... [**BLOCK FACTOR** *integer*]
... [{ **FULL** | **INCREMENTAL** | **INCREMENTAL SINCE FULL** }]
... [{ **VIRTUAL DECOUPLED** |
VIRTUAL ENCAPSULATED *'shell_command'* }]

...**TO** *archive_device* [**SIZE** *integer*] [**STACKER** *integer*] ...
... [**WITH COMMENT** *string*]

Examples

The following UNIX example backs up the asiqdemo database onto tape devices */dev/rmt/0* and */dev/rmt/2* on a Sun Solaris platform. On Solaris, the letter *n* after the device name specifies the "no rewind on close" feature. Always specify this feature with BACKUP , using the naming convention appropriate for your UNIX platform (Windows does not support this feature). This example backs up all changes to the database since the last full backup:

```
BACKUP DATABASE
INCREMENTAL SINCE FULL
TO '/dev/rmt/0n' SIZE 10000000
TO '/dev/rmt/2n' SIZE 15000000
```

Note Size units are kilobytes (KB). In this example, the specified sizes are 10GB and 15GB.

Usage

The IQ database might be open for use by many readers and writers when you execute a BACKUP command. It acts as a read-only user and relies on the Table Level Versioning feature of Sybase IQ to achieve a consistent set of data. BACKUP implicitly issues a CHECKPOINT prior to commencing, and then it backs up the catalog tables that describe the database (and any other tables you have added to the Catalog Store). During this first phase, Sybase IQ does not allow any metadata changes to the database (such as adding or dropping columns and tables). Correspondingly, a later RESTORE of the backup restores only up to that initial CHECKPOINT.

The BACKUP command lets you specify full or incremental backups. You can choose two kinds of incremental backups. INCREMENTAL backs up only those blocks that have changed and committed since the last BACKUP of any type (incremental or full). INCREMENTAL SINCE FULL backs up all of the blocks that have changed since the last full backup. The first type of incremental backup can be smaller and faster to do for BACKUP commands, but slower and more complicated for RESTORE commands. The opposite is true for the other type of incremental backup. The reason is that the first type generally results in *N* sets of incremental backup archives for each full backup archive. If a restore is required, the DBA must RESTORE the full backup archive first, and then each incremental archive in the proper order. (Sybase IQ keeps track of which ones are needed.) The second type requires the DBA to restore only the full backup archive and the last incremental archive.

Incremental virtual backup is supported using the VIRTUAL DECOUPLED and VIRTUAL ENCAPSULATED parameters of the BACKUP statement.

CRC clause Activates 32-bit cyclical redundancy checking on a per block basis (in addition to whatever error detection is available in the hardware). When you specify this clause, the numbers computed on backup are verified during any subsequent RESTORE operation, affecting performance of both commands. The default is ON.

ATTENDED clause Applies only when backing up to a tape device. If ATTENDED ON (the default) is used, a message is sent to the application that issued the BACKUP statement if the tape drive requires intervention. This might happen, for example, when a new tape is required. If you specify OFF, BACKUP does not prompt for new tapes. If additional tapes are needed and OFF has been specified, Sybase IQ gives an error and aborts the BACKUP command. However, a short delay is included to account for the time an automatic stacker drive requires to switch tapes.

BLOCK FACTOR clause Specifies the number of blocks to write at one time. Its value must be greater than 0, or Sybase IQ generates an error message. Its default is 25 for UNIX systems and 15 for Windows systems (to accommodate the smaller fixed tape block sizes). This clause effectively controls the amount of memory used for buffers. The actual amount of memory is this value times the block size times the number of threads used to extract data from the database. Sybase recommends setting BLOCK FACTOR to at least 25.

FULL clause Specifies a full backup; all blocks in use in the database are saved to the archive devices. This is the default action.

INCREMENTAL clause Specifies an incremental backup; all blocks changed since the last backup of any kind are saved to the archive devices.

INCREMENTAL SINCE FULL clause Specifies an incremental backup; all blocks changed since the last full backup are saved to the archive devices.

VIRTUAL DECOUPLED clause Specifies a decoupled virtual backup. For the backup to be complete, you must copy the IQ dbspaces after the decoupled virtual backup finishes, and then perform a nonvirtual incremental backup.

VIRTUAL ENCAPSULATED clause Specifies an encapsulated virtual backup. The *'shell-command'* argument can be a string or variable containing a string that is executed as part of the encapsulated virtual backup. The shell commands execute a system-level backup of the IQ Store as part of the backup operation.

TO clause Specifies the name of the archive_device to be used for backup, delimited with single quotation marks. The archive_device is a file name or tape drive device name for the archive file. If you are using multiple archive devices, specify them using separate TO clauses. (A comma-separated list is not allowed.) Archive devices must be distinct. The number of TO clauses determines the amount of parallelism Sybase IQ attempts with regard to output devices.

BACKUP overwrites existing archive files unless you move the old files or use a different *archive_device* name or path.

The backup API DLL implementation lets you specify arguments to pass to the DLL when opening an archive device. For third-party implementations, the *archive_device* string has the following format:

```
'DLLidentifier::vendor_specific_information'
```

A specific example:

```
'spsc::workorder=12;volname=ASD002'
```

The *archive_device* string length can be up to 1023 bytes. The *DLLidentifier* portion must be 1 to 30 bytes in length and can contain only alphanumeric and underscore characters. The *vendor_specific_information* portion of the string is passed to the third-party implementation without checking its contents. Do not specify the SIZE or STACKER clauses of the BACKUP command when using third-party implementations, as that information should be encoded in the *vendor_specific_information* portion of the string.

Note Only certain third-party products are certified with Sybase IQ using this syntax. See the *Sybase IQ Release Bulletin* for additional usage instructions or restrictions. Before using any third-party product to back up your Sybase IQ database in this way, make sure it is certified. See the *Sybase IQ Release Bulletin*, or see the Sybase Certification Reports for the Sybase IQ product in Technical Documents at http://www.sybase.com/support/techdocs/.

For the Sybase implementation of the backup API, you need to specify only the tape device name or file name. For disk devices, you should also specify the SIZE value, or Sybase IQ assumes that each created disk file is no larger than 2GB on UNIX, or 1.5GB on Windows. An example of an archive device for the Sybase API DLL that specifies a tape device for certain UNIX systems is:

```
'/dev/rmt/0'
```

SIZE clause Specifies maximum tape or file capacity per output device (some platforms do not reliably detect end-of-tape markers). No volume used on the corresponding device should be shorter than this value. This value applies to both tape and disk files but not third-party devices. Units are kilobytes (KB) so, for example, for a 3.5GB tape, you specify 3500000. Defaults are by platform and medium.

The SIZE parameter is per output device. SIZE does not limit the number of bytes per device; SIZE limits the file size. Each output device can have a different SIZE parameter. During backup, when the amount of information written to a given device reaches the value specified by the SIZE parameter, BACKUP does one of the following:

- If the device is a file system device, BACKUP closes the current file and creates another file of the same name, with the next ascending number appended to the file name, for example, *bkup1.dat1.1*, *bkup1.dat1.2*, *bkup1.dat1.3*.
- If the device is a tape unit, BACKUP closes the current tape and you need to mount another tape.

It is your responsibility to mount additional tapes if needed, or to ensure that the disk has enough space to accommodate the backup.

When multiple devices are specified, BACKUP distributes the information across all devices.

Table 6-1: BACKUP default sizes

Platform	Default SIZE for tape	Default SIZE for disk
UNIX	none	2GB
Windows	1.5GB SIZE must be a multiple of 64. Other values are rounded down to a multiple of 64.	1.5GB

STACKER clause pecifies that the device is automatically loaded, and specifies the number of tapes with which it is loaded. This value is not the tape position in the stacker, which could be zero. When ATTENDED is OFF and STACKER is ON, Sybase IQ waits for a predetermined amount of time to allow the next tape to be autoloaded. The number of tapes supplied along with the SIZE clause are used to determine whether there is enough space to store the backed-up data. Do not use this clause with third-party media management devices.

WITH COMMENT clause Specifies an optional comment recorded in the archive file and in the backup history file. Maximum length is 32KB. If you do not specify a value, a NULL string is stored.

Other issues for BACKUP include:

- BACKUP does not support raw devices as archival devices.
- Windows systems support only fixed-length I/O operations to tape devices (for more information about this limitation, see your *Sybase IQ Installation and Configuration Guide*). Although Windows supports tape partitioning, Sybase IQ does not use it, so do not use another application to format tapes for BACKUP. Windows has a simpler naming strategy for its tape devices, where the first tape device is \\.*tape0*, the second is \\.*tape1*, and so on.

  Warning! For backup (and for most other situations) Sybase IQ treats the leading backslash in a string as an escape character, when the backslash precedes an n, an x, or another backslash. For this reason, when you specify backup tape devices, you must double each backslash required by the Windows naming convention. For example, indicate the first Windows tape device you are backing up to as '\\\\.*tape0*', the second as '\\\\.*tape1*', and so on. If you omit the extra backslashes, or otherwise misspell a tape device name, and write a name that is not a valid tape device on your system, Sybase IQ interprets this name as a disk file name.

- Sybase IQ does not rewind tapes before using them. You must ensure the tapes used for BACKUP or RESTORE are at the correct starting point before putting them in the tape device. Sybase IQ does rewind tapes after using them on rewinding devices.
- During BACKUP and RESTORE operations, if Sybase IQ cannot open the archive device (for example, when it needs the media loaded) and the ATTENDED parameter is ON, it waits for ten seconds and tries again. It continues these attempts indefinitely until either it is successful or the operation is terminated with a Ctrl+C.
- If you enter Ctrl+C, BACKUP fails and returns the database to the state it was in before the backup started.
- If disk striping is used, such as on a RAID device, the striped disks are treated as a single device.
- If you are recovering an Adaptive Server Anywhere database, see "Backup and Data Recovery" in *Adaptive Server Anywhere Database Administration Guide* for additional options.

Side effects

Automatic commit.

Standards

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

Permissions

Must be the owner of the database or have DBA authority.

See also

RESTORE statement on page 621

Chapter 14, "Data Backup, Recovery, and Archiving," in *Sybase IQ System Administration Guide*

BEGIN... END statement

Description

Groups SQL statements together.

Syntax

[*statement-label* :]
... **BEGIN** [[**NOT**] **ATOMIC**]
... [*local-declaration* ; ...]
... *statement-list*
... [**EXCEPTION** [*exception-case* ...]]
... **END** [*statement-label*]

Parameters

local-declaration:
{ *variable-declaration*
| *cursor-declaration*
| *exception-declaration*
| *temporary-table-declaration* }

variable-declaration:
DECLARE *variable-name data-type*

exception-declaration:
DECLARE *exception-name* EXCEPTION
FOR
SQLSTATE [VALUE] *string*

exception-case:
WHEN *exception-name* [, ...] THEN *statement-list*
| WHEN OTHERS THEN *statement-list*

Examples

The body of a procedure is a compound statement:

```
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35),
OUT TopValue INT)
```

```
BEGIN
    DECLARE err_notfound EXCEPTION FOR
       SQLSTATE '02000' ;
    DECLARE curThisCust CURSOR FOR
       SELECT company_name, CAST(
              sum(sales_order_items.quantity *
              product.unit_price) AS INTEGER) VALUE
       FROM customer
              LEFT OUTER JOIN sales_order
              LEFT OUTER JOIN sales_order_items
              LEFT OUTER JOIN product
       GROUP BY company_name ;
    DECLARE ThisValue INT ;
    DECLARE ThisCompany CHAR(35) ;
    SET TopValue = 0 ;
    OPEN curThisCust ;


    CustomerLoop:
    LOOP
       FETCH NEXT curThisCust
          INTO ThisCompany, ThisValue ;
       IF SQLSTATE = err_notfound THEN
          LEAVE CustomerLoop ;
       END IF ;
       IF ThisValue > TopValue THEN
          SET TopValue = ThisValue ;
          SET TopCompany = ThisCompany ;
       END IF ;
    END LOOP CustomerLoop ;


CLOSE curThisCust ;
END
```

Usage

The body of a procedure or trigger is a **compound statement**. Compound statements can also be used in control statements within a procedure or trigger.

A compound statement allows one or more SQL statements to be grouped together and treated as a unit. A compound statement starts with BEGIN and ends with END. Immediately following BEGIN, a compound statement can have local declarations that exist only within the compound statement. A compound statement can have a local declaration for a variable, a cursor, a temporary table, or an exception. Local declarations can be referenced by any statement in that compound statement, or in any compound statement nested within it. Local declarations are not visible to other procedures that are called from within a compound statement.

If the ending *statement-label* is specified, it must match the beginning *statement-label*. The LEAVE statement can be used to resume execution at the first statement after the compound statement. The compound statement that is the body of a procedure has an implicit label that is the same as the name of the procedure or trigger.

For a complete description of compound statements and exception handling, see Chapter 8, "Using Procedures and Batches" in the *Sybase IQ System Administration Guide*.

Side effects

None.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** Supported by Adaptive Server Enterprise. This does not mean that all statements inside a compound statement are supported.

  BEGIN and END keywords are not required in Transact-SQL.

  BEGIN and END are used in Transact-SQL to group a set of statements into a single compound statement, so that control statements such as IF ... ELSE, which affect the performance of only a single SQL statement, can affect the performance of the whole group. The ATOMIC keyword is not supported by Adaptive Server Enterprise.

  In Transact-SQL. DECLARE statements need not immediately follow BEGIN, and the cursor or variable that is declared exists for the duration of the compound statement. You should declare variables at the beginning of the compound statement for compatibility.

Permissions

None

See also

DECLARE LOCAL TEMPORARY TABLE statement on page 523

DECLARE CURSOR statement [ESQL] [SP] on page 516

LEAVE statement on page 578

RESIGNAL statement on page 620

SIGNAL statement on page 652

BEGIN PARALLEL IQ ... END PARALLEL IQ statement

Description

Groups CREATE INDEX statements together for execution at the same time.

Syntax

... **BEGIN PARALLEL IQ**
statement-list
... **END PARALLEL IQ**

Parameters

statement-list
a list of CREATE INDEX statements

Examples

The following statement executes atomically. If one command fails, the entire statement rolls back:

```
BEGIN PARALLEL IQ
       CREATE HG INDEX c1_HG on table1 (col1);
       CREATE HNG INDEX c12_HNG on table1 (col12);
       CREATE LF INDEX c1_LF on table1 (col1);
       CREATE HNG INDEX c2_HNG on table1 (col2);
END PARALLEL IQ
```

Usage

The BEGIN PARALLEL IQ ... END PARALLEL IQ statement lets you execute a group of CREATE INDEX statements as though they are a single DDL statement, creating indexes on multiple IQ tables at the same time. While this statement is executing, you and other users cannot issue other DDL statements.

You can specify multiple tables within the statement list. Granularity is at the column level. In other words, multiple indexes on the same column are executed serially.

Side effects

Automatic commit.

Standards

- **SQL92** Not supported.
- **Sybase** Not supported by Adaptive Server Enterprise. For support of statements inside the statement, see CREATE INDEX statement on page 473.

Permissions

None

See also

CREATE INDEX statement on page 473

BEGIN TRANSACTION statement

Description

Starts a user-defined transaction.

Syntax

BEGIN TRAN[**SACTION**] [*transaction-name*]

Examples

Example 1 Illustrates the effect of a BEGIN TRANSACTION statement on the snapshot version of a table:

In the first case, assume that table t1 contains no data. Two connections, Conn1 and Conn2, are made at the same time. Table 6-2 is a timeline of the commands executed within the two connections:

Table 6-2: first case command timeline

Conn1	Conn2
`CONNECT`	`CONNECT`
`INSERT t1 VALUES(1)` (an implicit begin transaction)	…
`COMMIT`	…
…	`SELECT * FROM t1` (an implicit begin transaction)
	Data returned from table t1: 1

In the first case, user Conn2 issues a SELECT statement after user Conn1 issues a COMMIT. Since the SELECT of Conn2 is the first command executed following the connect, a transaction begins at this time and a snapshot is taken of table t1 after t1 contains data. User Conn2 can see the updated table.

In the second case, assume again that table t1 contains no data. Two connections, Conn1 and Conn2, are made at the same time. The commands executed by the two users are in the following timeline:

Table 6-3: second command timeline

Conn1	Conn2
CONNECT	CONNECT
…	BEGIN TRANSACTION
INSERT t1 VALUES(1) (an implicit begin transaction)	…
COMMIT	…
…	SELECT * FROM t1
	No data returned from table t1

In this case, user Conn2 issues a BEGIN TRANSACTION statement after connecting and Sybase IQ takes a snapshot of table t1 before user Conn1 inserts any data. Even though Conn2 issues a SELECT after Conn1 has committed the inserted data, Conn2 still has a snapshot of t1 *before* the data was inserted. In this case, Conn2 cannot see the updated table and the SELECT returns no data. Until the current transaction of user Conn2 ends, the image of table t1 remains unchanged to user Conn2.

Example 2 The following batch reports successive values of @@trancount as 0, 1, 2, 1, 0. The values are printed on the server window:

```
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
```

See "Usage," below, for more information about the @@trancount global variable.

Usage

The optional parameter *transaction-name* is the name assigned to this transaction. It must be a valid identifier. Use transaction names only on the outermost pair of nested BEGIN/COMMIT or BEGIN/ROLLBACK statements.

BEGIN TRANSACTION creates a transaction for the current connection, if the connection does not currently have a transaction. When a transaction starts, it selects the snapshot version that is used until the next commit or rollback.

A transaction automatically starts at the start of the first command following a connect, commit, or rollback, if there is no explicit BEGIN TRANSACTION.

When executed inside a transaction, BEGIN TRANSACTION increases the nesting level of transactions by one. The nesting level is decreased by a COMMIT statement. When transactions are nested, only the outermost COMMIT makes the changes to the database permanent.

Chained and unchained modes

Adaptive Server Enterprise and Sybase IQ have two transaction modes.

The default Adaptive Server Enterprise transaction mode, called **unchained mode**, commits each statement individually, unless an explicit BEGIN TRANSACTION statement is executed to start a transaction. In contrast, the SQL92-compatible **chained mode** commits a transaction only when an explicit COMMIT is executed, or when a statement that carries out an autocommit (such as data definition statements) is executed.

You can control the mode by setting the CHAINED database option. The default setting for ODBC and Embedded SQL connections in Sybase IQ is ON, in which case Sybase IQ runs in chained mode. (ODBC users should also check the AutoCommit ODBC setting.) The default for TDS connections is OFF.

You cannot alter the CHAINED option within a transaction.

Warning! When calling a stored procedure, ensure that it operates correctly under the required transaction mode.

For more information about the CHAINED option and the chained mode, see “CHAINED option [TSQL]” on page 51.

The current nesting level is held in the global variable @@trancount. The @@trancount variable has a value of zero before a BEGIN TRANSACTION statement is executed, and only a COMMIT executed when @@trancount is equal to one makes changes to the database permanent.

A ROLLBACK statement without a transaction or savepoint name always rolls back statements to the outermost BEGIN TRANSACTION (explicit or implicit) statement, and cancels the entire transaction.

@@trancount values in Adaptive Server Enterprise and IQ

Do not rely on the value of @@trancount for more than keeping track of the number of explicit BEGIN TRANSACTION statements that have been issued.

When Adaptive Server Enterprise starts a transaction implicitly, @@trancount is set to 1. Sybase IQ does not set the @@trancount value to 1 when a transaction is started implicitly. Consequently, the IQ @@trancount variable has a value of zero before any BEGIN TRANSACTION statement (even though there is a current transaction), while in Adaptive Server Enterprise (in chained mode) @@trancount has a value of 1.

For transactions starting with a BEGIN TRANSACTION statement, @@trancount has a value of 1 in both Sybase IQ and Adaptive Server Enterprise after the BEGIN TRANSACTION statement. If a transaction is started implicitly with a different statement, and a BEGIN TRANSACTION statement is then executed, @@trancount has a value of 2 in both Sybase IQ and Adaptive Server Enterprise after the BEGIN TRANSACTION statement.

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** Supported by Adaptive Server Enterprise.

Permissions

None.

See also

"ISOLATION_LEVEL option" on page 93

COMMIT statement on page 436

ROLLBACK statement on page 630

SAVEPOINT statement on page 632

CALL statement

Description

Invokes a procedure.

Syntax

Syntax 1

[*variable* =] **CALL** *procedure-name* ([*expression*] [, ...])

Syntax 2

[*variable* =] **CALL** *procedure-name* ([*parameter-name* = *expression*] [, ...])

Examples

- This example calls the sp_customer_list procedure. This procedure has no parameters, and returns a result set:

```
CALL sp_customer_list()
```

- This DBISQL example creates a procedure to return the number of orders placed by the customer whose ID is supplied, creates a variable to hold the result, calls the procedure, and displays the result:

```
CREATE PROCEDURE OrderCount (IN customer_ID INT, OUT
Orders INT)
BEGIN
SELECT COUNT("DBA".sales_order.id)
INTO Orders
FROM "DBA".customer
KEY LEFT OUTER JOIN "DBA".sales_order
WHERE "DBA".customer.id = customer_ID ;
END
go
-- Create a variable to hold the result
CREATE VARIABLE Orders INT
go

-- Call the procedure, FOR customer 101
-- ----------------------------
CALL OrderCount ( 101, Orders)
go
--------------------------------
--  Display the result
SELECT Orders FROM DUMMY
go
```

Usage

CALL invokes a procedure that has been previously created with a CREATE PROCEDURE statement. When the procedure completes, any INOUT or OUT parameter values are copied back.

You can specify the argument list by position or by using keyword format. By position, arguments match up with the corresponding parameter in the parameter list for the procedure. By keyword, arguments match the named parameters.

Procedure arguments can be assigned default values in the CREATE PROCEDURE statement, and missing parameters are assigned the default value, or, if no default is set, NULL.

Inside a procedure, CALL can be used in a DECLARE statement when the procedure returns result sets (see Chapter 8, “Using Procedures and Batches” in the *Sybase IQ System Administration Guide*).

Procedures can return an integer value (as a status indicator, say) using the RETURN statement. You can save this return value in a variable using the equality sign as an assignment operator:

```
CREATE VARIABLE returnval INT ;
returnval = CALL proc_integer ( arg1 = val1, ... )
```

Side effects

None.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** Not supported by Adaptive Server Enterprise. For an alternative that is supported, see EXECUTE statement [ESQL] on page 541.

Permissions

Must be the owner of the procedure, have EXECUTE permission for the procedure, or have DBA authority.

See also

CREATE PROCEDURE statement on page 485

GRANT statement on page 559

CASE statement

Description

Selects execution path based on multiple cases.

Syntax

CASE *value-expression*
... **WHEN** [*constant* | **NULL**] **THEN** *statement-list* ...
... [**WHEN** [*constant* | **NULL**] **THEN** *statement-list*] ...
... **ELSE** *statement-list*
... **END CASE**

Examples

This procedure using a CASE statement classifies the products listed in the product table of the sample database into one of shirt, hat, shorts, or unknown:

```
CREATE PROCEDURE ProductType (IN product_id INT, OUT
type CHAR(10))
    BEGIN
    DECLARE prod_name CHAR(20) ;
    SELECT name INTO prod_name FROM "DBA"."product"
    WHERE id = product_id;
    CASE prod_name
    WHEN 'Tee Shirt' THEN
       SET type = 'Shirt'
    WHEN 'Sweatshirt' THEN
       SET type = 'Shirt'
    WHEN 'Baseball Cap' THEN
       SET type = 'Hat'
    WHEN 'Visor' THEN
       SET type = 'Hat'
```

```
WHEN 'Shorts' THEN
   SET type = 'Shorts'
ELSE
   SET type = 'UNKNOWN'
END CASE ;
END
```

Usage

The CASE statement is a control statement that lets you choose a list of SQL statements to execute based on the value of an expression. If a WHEN clause exists for the value of *value-expression*, the *statement-list* in the WHEN clause is executed. If no appropriate WHEN clause exists, and an ELSE clause exists, the *statement-list* in the ELSE clause is executed. Execution resumes at the first statement after the END CASE.

Note The ANSI standard allows two forms of CASE statements. Although Sybase IQ allows both forms, when CASE is in the predicate, for best performance you must use the form shown here.

If you require the other form (also called ANSI syntax) for compatibility with Adaptive Server Anywhere, see CASE statement Syntax 2 in *Adaptive Server Anywhere SQL Reference*.

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

For information on the CASE expression, see “Expressions” on page 179.

Side effects

None.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** Not supported by Adaptive Server Enterprise.

Permissions

None.

See also

BEGIN... END statement on page 422

CHECKPOINT statement

Description: Checkpoints the database.

Syntax: **CHECKPOINT**

Usage: CHECKPOINT forces the database server to execute a checkpoint. Checkpoints are also performed automatically by the database server according to an internal algorithm. Applications do not normally need to issue CHECKPOINT. For a full description of checkpoints, see Chapter 14, "Data Backup, Recovery, and Archiving" in the *Sybase IQ System Administration Guide*.

Side effects

None.

Standards:

- **SQL92** Vendor extension
- **Sybase** Supported by Adaptive Server Enterprise.

Permissions: Must have DBA authority to checkpoint the network database server. No permissions are required to checkpoint the personal database server.

CLEAR statement [DBISQL]

Description: Clears the Interactive SQL (DBISQL) data window.

Syntax: **CLEAR**

Side Effects

Closes the cursor associated with the data being cleared.

Usage: The CLEAR statement is used to clear the DBISQL main window.

Side effects

Closes the cursor associated with the data being cleared.

Standards:

- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions: None.

See also: EXIT statement [DBISQL] on page 546

CLOSE statement [ESQL] [SP]

Description: Closes a cursor.

Syntax: **CLOSE** *cursor-name*

Parameters: *cursor-name*:
{ *identifier* | *host-variable* }

Examples

- Close cursors in Embedded SQL:

```
EXEC SQL CLOSE employee_cursor;
EXEC SQL CLOSE :cursor_var;
```

- Uses a cursor:

```
CREATE PROCEDURE TopCustomer (OUT TopCompany
CHAR(35), OUT TopValue INT)
BEGIN
DECLARE err_notfound EXCEPTION
   FOR SQLSTATE '02000' ;
DECLARE curThisCust CURSOR FOR
SELECT company_name, CAST(
sum(sales_order_items.quantity *
product.unit_price) AS INTEGER) VALUE
FROM customer
LEFT OUTER JOIN sales_order
LEFT OUTER JOIN sales_order_items
LEFT OUTER JOIN product
GROUP BY company_name ;


DECLARE ThisValue INT ;
DECLARE ThisCompany CHAR(35) ;
SET TopValue = 0 ;
OPEN curThisCust ;
CustomerLoop:
LOOP
   FETCH NEXT curThisCust
   INTO ThisCompany, ThisValue ;
      IF SQLSTATE = err_notfound THEN
         LEAVE CustomerLoop ;
      END IF ;
      IF ThisValue > TopValue THEN
         SET TopValue = ThisValue ;
         SET TopCompany = ThisCompany ;
      END IF ;
   END LOOP CustomerLoop ;
CLOSE curThisCust ;
```

```
END
```

Usage

This statement closes the named cursor.

Side effects

None.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Adaptive Server Enterprise.

Permissions

The cursor must have been previously opened.

See also

DECLARE CURSOR statement [ESQL] [SP] on page 516

OPEN statement [ESQL] [SP] on page 603

PREPARE statement [ESQL] on page 611

COMMENT statement

Description

Stores a comment in the system tables for a database object.

Syntax

COMMENT ON
{ **COLUMN** [*owner*.]*table-name.column-name*
| **EVENT** *event-name*
| **FOREIGN KEY** [*owner*.]*table-name.role-name*
| **INDEX** [[*owner*.]*table*.]*index-name*
| **JAVA CLASS** *java-class-name*
|**JAVA JAR** *java-jar-name*
| **LOGIN** *integrated_login_id*
| **PROCEDURE** [*owner*.]*procedure-name*
| **SERVICE** *web-service-name*
| **TABLE** [*owner*.]*table-name*
| **USER** *userid*
| **VIEW** [*owner*.]*view-name* }
IS *comment*

Parameters

comment:
{ *string* | NULL }

Examples

These examples show how to add and remove a comment.

- Adds a comment to the employee table:

```
COMMENT
ON TABLE employee
IS "Employee information"
```

- Removes the comment from the employee table:

```
COMMENT
ON TABLE employee
IS NULL
```

Usage

Several system tables have a column named Remarks that lets you associate a comment with a database item:

Table 6-4: System tables with Remarks column

SYSCOLUMN	SYSLOGIN
SYSEVENT	SYSPROCEDURE
SYSFOREIGNKEY	SYSPROCPARM
SYSINDEX	SYSPUBLICATION
SYSIQJOININDEX	SYSREMOTETYPE
SYSJAR	SYSTABLE
SYSJARCOMPONENT	SYSUSERPERM
SYSJAVACLASS	

COMMENT ON lets you set the Remarks column in these system tables. You can remove a comment by setting it to NULL.

Side effects

Automatic commit.

Standards

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

Permissions

Must either be the owner of the database object being commented, or have DBA authority.

COMMIT statement

Description

Makes changes to the database permanent, or terminates a user-defined transaction.

Syntax

Syntax 1

COMMIT [**WORK**]

Syntax 2

COMMIT TRAN[**SACTION**] [*transaction-name*]

Examples

- This statement commits the current transaction:

```
COMMIT
```

- The following Transact-SQL batch reports successive values of @@trancount as 0, 1, 2, 1, 0:

```
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
go
```

Usage

Syntax 1 The COMMIT statement ends a transaction and makes all changes made during this transaction permanent in the database.

Data definition statements carry out commits automatically. For information, see the Side effects listing for each SQL statement.

COMMIT fails if the database server detects any invalid foreign keys. This makes it impossible to end a transaction with any invalid foreign keys. Usually, foreign key integrity is checked on each data manipulation operation. However, if the database option WAIT_FOR_COMMIT is set ON or a particular foreign key was defined with a CHECK ON COMMIT clause, the database server delays integrity checking until the COMMIT statement is executed.

Syntax 2 You can use BEGIN TRANSACTION and COMMIT TRANSACTION statements in pairs to construct **nested transactions**. Nested transactions are similar to **savepoints**. When executed as the outermost of a set of nested transactions, the statement makes changes to the database permanent. When executed inside a transaction, COMMIT TRANSACTION decreases the nesting level of transactions by one. When transactions are nested, only the outermost COMMIT makes the changes to the database permanent.

The optional parameter *transaction-name* is the name assigned to this transaction. It must be a valid identifier. Use transaction names only on the outermost pair of nested BEGIN/COMMIT or BEGIN/ROLLBACK statements.

You can use a set of options to control the detailed behavior of the COMMIT statement. For information, see "COOPERATIVE_COMMIT_TIMEOUT option" on page 61, "COOPERATIVE_COMMITS option" on page 61, "DELAYED_COMMITS option" on page 72, and "DELAYED_COMMIT_TIMEOUT option" on page 72. You can use the Commit connection property to return the number of commits on the current connection.

Side effects

Closes all cursors except those opened WITH HOLD.

Deletes all rows of declared temporary tables on this connection, unless they were declared using ON COMMIT PRESERVE ROWS.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Adaptive Server Enterprise. Syntax 2 is a Transact-SQL extension.

Permissions

Must be connected to the database.

See also

BEGIN TRANSACTION statement on page 426

CONNECT statement [ESQL] [DBISQL] on page 439

DISCONNECT statement [DBISQL] on page 532

ROLLBACK statement on page 630

SAVEPOINT statement on page 632

SET CONNECTION statement [DBISQL] [ESQL] on page 645

CONFIGURE statement [DBISQL]

Description

Activates the DBISQL configuration window.

Syntax

CONFIGURE

Usage

The DBISQL configuration window displays the current settings of all DBISQL options. It does not display or let you modify database options.

If you select Permanent, the options are written to the SYSOPTION table in the database and the database server performs an automatic COMMIT. If you do not choose Permanent, and instead click OK, options are set temporarily and remain in effect for the current database connection only.

Side effects

None.

Standards

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

Permissions

None.

See also

SET OPTION statement on page 647

CONNECT statement [ESQL] [DBISQL]

Description

Establishes a connection to a database.

Syntax

Syntax 1

CONNECT
... [**TO** *engine-name*]
... [**DATABASE** *database-name*]
... [**AS** *connection-name*]
... [**USER**] *userid* [**IDENTIFIED BY** *password*]

Syntax 2

CONNECT USING *connect-string*

Parameters

engine-name:
identifier, string, or host-variable

database-name:
identifier, string, or host-variable

connection-name:
identifier, string, or host-variable

userid:
identifier, string, or host-variable

password:
identifier, strin,g or host-variable

connect-string:
a valid connection string or host-variable

Examples

These are examples of CONNECT usage within Embedded SQL:

```
EXEC SQL CONNECT AS :conn_name
USER :userid IDENTIFIED BY :password;
```

```
EXEC SQL CONNECT USER "dba" IDENTIFIED BY "SQL";
```

These are examples of CONNECT usage from DBISQL.

- Connect to a database from DBISQL. Prompts display for user ID and password:

  ```
  CONNECT
  ```

- Connect to the default database as DBA, from DBISQL. A password prompt displays:

  ```
  CONNECT USER "DBA"
  ```

- Connect to the sample database as the DBA, from DBISQL:

  ```
  CONNECT
  TO asiqdemo
  USER "DBA"
  IDENTIFIED BY SQL
  ```

- Connect to the sample database using a connect string, from DBISQL:

  ```
  CONNECT
  USING 'UID=DBA;PWD=SQL;DBN=asiqdemo'
  ```

Usage

The CONNECT statement establishes a connection to the database identified by *database-name* running on the server identified by *engine-name*.

Embedded SQL behavior In Embedded SQL, if no *engine-name* is specified, the default local database server is assumed (the first database server started). If a local database server is not running and the Anywhere Client (DBCLIENT) is running, the default server is assumed (the server name specified when the client was started). If no *database-name* is specified, the first database on the given server is assumed.

The WHENEVER statement, SET SQLCA, and some DECLARE statements do not generate code and thus might appear before the CONNECT statement in the source file. Otherwise, no statements are allowed until a successful CONNECT statement has been executed.

The user ID and password are used for permission checks on all dynamic SQL statements. By default, the password is case sensitive; the user ID is not.

For a detailed description of the connection algorithm, see “How Sybase IQ makes connections” in Chapter 3, “Sybase IQ Connections” in the *Sybase IQ System Administration Guide*.

DBISQL behavior If no database or server is specified in the CONNECT statement, DBISQL remains connected to the current database, rather than to the default server and database. If a database name is specified without a server name, DBISQL attempts to connect to the specified database on the current server. You must specify the database name defined in the -n database switch, not the database file name. If a server name is specified without a database name, DBISQL connects to the default database on the specified server. For example, if the following batch is executed while connected to a database, the two tables are created in the same database.

```
CREATE TABLE t1( c1 int );
CONNECT DBA IDENTIFIED BY SQL;
CREATE TABLE t2 (c1 int );
```

No other database statements are allowed until a successful CONNECT statement has been executed.

The user ID and password are used for checking the permissions on SQL statements. If the password or the user ID and password are not specified, the user is prompted to type the missing information. By default, the password is case sensitive; the user ID is not.

Multiple connections are managed through the concept of a current connection. After a successful connect statement, the new connection becomes the current one. To switch to a different connection, use SET CONNECTION. Executing a CONNECT statement does not close the existing connection (if any). Use DISCONNECT to drop connections.

Static SQL statements use the user ID and password specified with the *-l* option on the SQLPP statement line. If no *-l* option is given, then the user ID and password of the CONNECT statement are used for static SQL statements also.

Connecting with no password If you are connected to a user ID with DBA authority, you can connect to another user ID without specifying a password. (The output of dbtran requires this capability.) For example, if you are connected to a database from Interactive SQL as DBA, you can connect without a password with the statement:

```
CONNECT other_user_id
```

In Embedded SQL, you can connect without a password by using a host variable for the password and setting the value of the host variable to be the null pointer.

AS clause A connection can optionally be named by specifying the AS clause. This allows multiple connections to the same database, or multiple connections to the same or different database servers, all simultaneously. Each connection has its own associated transaction. You might even get locking conflicts between your transactions if, for example, you try to modify the same record in the same database from two different connections.

Syntax 2 A *connect-string* is a list of parameter settings of the form keyword=*value*, and must be enclosed in single quotes.

Side effects

None.

Standards

- **SQL92** Syntax 1 is a full SQL feature; Syntax 2 is a vendor extension.
- **Sybase** Open Client Embedded SQL supports a different syntax for the CONNECT statement.

Permissions

None.

See also

DISCONNECT statement [DBISQL] on page 532

GRANT statement on page 559

SET CONNECTION statement [DBISQL] [ESQL] on page 645

CREATE DATABASE statement

Description

Creates a database consisting of several operating system files.

Syntax

CREATE DATABASE *db-name*
... [[**TRANSACTION**] { **LOG ON** [*log-file-name*]
[**MIRROR** *mirror-file-name*] }]
... [**CASE** { **RESPECT** | **IGNORE** }]
... [**PAGE SIZE** *page-size*]
... [**COLLATION** *collation-label*] [**ENCRYPTED**
ON | **OFF** | *key-spec*}] {... [**BLANK PADDING ON**]
]
... [**JAVA** { **ON** | **OFF** }]
... [**JCONNECT** { **ON** | **OFF** }]
... [**PASSWORD CASE** { **RESPECT** | **IGNORE** }]
... [**IQ PATH** *iq-file-name*]
... [**IQ SIZE** *iq-file-size*]
... [**IQ PAGE SIZE** *iq-page-size*]
... [**BLOCK SIZE** *block-size*]
... [**IQ RESERVE** *sizeMB*]
... [**TEMPORARY RESERVE** *sizeMB*]

... [**MESSAGE PATH** *message-file-name*]
... [**TEMPORARY PATH** *temp-file-name*]
.... [**TEMPORARY SIZE** *temp-db-size*]

Parameters

db-name | *log-file-name* | *mirror-file-name* | *iq-file-name* | *message-file-name* | *temp-file-name*:
'*file-name*'

page-size:
{ 4096 | 8192 | 16384 | 32768 }

iq-page-size:
{ 65536 | 131072 | 262144 | 524288 }

block-size:
{ 4096 | 8192 | 16384 | 32768 }

collation-label:
string

key-spec: [ON] KEY *key* [ALGORITHM 'AES']

Examples

- The following Windows example creates an IQ database named *mydb* with its corresponding *mydb.db*, *mydb.iq*, *mydb.iqtmp*, and *mydb.iqmsg* files in the *C:\s1\data* directory:

```
CREATE DATABASE 'C:\\s1\\data\\mydb'
BLANK PADDING ON
IQ PATH 'C:\\s1\\data'
IQ SIZE 2000
IQ PAGE SIZE 65536
```

- The following UNIX command creates an IQ database with raw devices for IQ PATH and TEMPORARY PATH. The default IQ page size of 128KB applies.

```
CREATE DATABASE '/s1/data/bigdb'
IQ PATH '/dev/md/rdsk/bigdb'
MESSAGE PATH '/s1/data'
TEMPORARY PATH '/dev/md/rdsk/bigtmp'
```

- The following Windows command creates an IQ database with a raw device for IQ PATH. Note the doubled backslashes in the raw device name (a Windows requirement):

```
CREATE DATABASE 'company'
IQ PATH '\\\\.\\E:'
JCONNECT OFF
IQ SIZE 40
```

- The following UNIX example creates a strongly encrypted IQ database using the AES encryption algorithm with the key “is!seCret.”

```
CREATE DATABASE 'marvin.db'
JAVA OFF
BLANK PADDING ON
CASE RESPECT
COLLATION 'ISO_BINENG'
IQ PATH '/filesystem/marvin.main1'
IQ SIZE 6400
IQ PAGE SIZE 262144
TEMPORARY PATH '/filesystem/marvin.temp1'
TEMPORARY SIZE 3200
MESSAGE PATH '/filesystem/marvin.mess1'
ENCRYPTED ON KEY 'is!seCret' ALGORITHM 'AES'
```

Usage

Creates an IQ database with the supplied name and attributes. The IQ PATH clause is required for creating the IQ database. Otherwise, you create a standard Adaptive Server Anywhere database. If you omit the IQ PATH option, specifying any of the following options generates an error: IQ SIZE, IQ PAGE SIZE, BLOCK SIZE, MESSAGE PATH, TEMPORARY PATH, and TEMPORARY SIZE.

When Sybase IQ creates an IQ database, it automatically generates four database files to store different types of data that constitute an IQ database. Each file corresponds to a dbspace, the logical name by which Sybase IQ identifies database files. The files are:

- *db-name.db* is the file that holds the catalog dbspace, SYSTEM. It contains the system tables and stored procedures describing the database and any standard Anywhere database objects you add. If you do not include the *.db* extension, Sybase IQ adds it. This initial dbspace contains the Catalog Store, and you can later add dbspaces to increase its size. It cannot be created on a raw partition.
- *db-name.iq* is the default name of the file that holds the main data dbspace, IQ_SYSTEM_MAIN, containing the IQ tables and indexes. You can specify a different file name with the IQ PATH clause. This initial dbspace contains the IQ Store, and you can later add dbspaces to increase its size.
- *db-name.iqtmp* is the default name of the file that holds the initial temporary dbspace, IQ_SYSTEM_TEMP. It contains the temporary tables generated by certain queries. The required size of this file can vary depending on the type of query and amount of data. You can specify a different name using the TEMPORARY PATH clause. This initial dbspace contains the Temporary Store, and you can later add dbspaces to increase its size.

- *db-name.iqmsg* is the default name of the file that contains the messages trace dbspace, IQ_SYSTEM_MSG. You can specify a different file name using the MESSAGE PATH clause.

In addition to these files, an IQ database has a transaction log file (*db-name.log*), and might have a transaction log mirror file.

File names

The file names (*db-name*, *log-file-name*, *mirror-file-name*, *iq-file-name*, *message-file-name*, *temp-file-name*) are strings containing operating system file names. As literal strings, they must be enclosed in single quotes.

- In Windows, if you specify a path, any backslash characters (\) must be doubled if they are followed by an n or an x. This prevents them being interpreted as a newline character (\n) or as a hexadecimal number (\x), according to the rules for strings in SQL. It is safer to always double the backslash. For example:

```
CREATE DATABASE 'c:\\sybase\\mydb.db'
LOG ON 'e:\\logdrive\\mydb.log'
JCONNECT OFF
IQ PATH 'c:\\sybase\\mydb'
IQ SIZE 40
```

- If you specify no path, or a relative path:
  - The Catalog Store file (*db-name.db*) is created relative to the working directory of the server.
  - The IQ Store, Temporary Store, and message log files are created in the same directory as, or relative to, the Catalog Store.

Relative path names are recommended.

Warning! The database file, temporary dbspace, and transaction log file *must* be located on the same physical machine as the database server. Do not place database files and transaction log files on a network drive. The transaction log should be on a separate device from its mirror, however.

On UNIX systems, you can create symbolic links, which are indirect pointers that contain the path name of the file to which they point. You can use symbolic links as relative path names. There are several advantages to creating a symbolic link for the database file name:

- Symbolic links to raw devices can have meaningful names, while the actual device name syntax can be obscure.

- A symbolic name might eliminate problems restoring a database file that was moved to a new directory since it was backed up.
- In multiplex databases, symbolic links can be used to avoid device name conflicts among multiple servers when you use raw devices for IQ Temporary storage.

To create a symbolic link, use the ln -s command. For example:

```
ln -s /disk1/company/iqdata/company.iq company_iq_store
```

Once you create this link, you can specify the symbolic link in commands like CREATE DATABASE or RESTORE instead of the fully qualified path name.

When you create a database or a dbspace, the path for every dbspace file must be unique. If your CREATE DATABASE command specifies the identical path and file name for these two stores, you receive an error.

Note Multiplex databases have a shared IQ Store, where they share all dbspaces, and a local IQ Store. The local IQ Store consists of dbspaces that are managed by only one query server and are not visible to any other query server. To create multiplex databases, use the Create Database and Create Query Server wizards in Sybase Central. See Chapter 5, "Working with Database Objects" in the *Sybase IQ System Administration Guide* for more information.

You can create a unique path in any of these ways:

- Specify a different extension for each file (for example, *mydb.iq* and *mydb.iqtmp*)
- Specify a different file name (for example, *mydb.iq* and *mytmp.iq*)
- Specify a different path name (for example, */iqfiles/main/iq* and */iqfiles/temp/iq*) or different raw partitions
- Omit TEMPORARY PATH when you create the database. In this case, the temporary store is created in the same path as the Catalog Store, with the default name and extension *dbname.iqtmp*, where *dbname* is the database name.

Warning! On UNIX platforms, to maintain database consistency, you must specify file names that are links to different files. Sybase IQ cannot detect the target where linked files point. Even if the file names in the command differ, it is your responsibility to make sure they do not point to the same file.

Clauses and options of CREATE DATABASE

TRANSACTION LOG The transaction log is a file where the database server logs all changes made to the database. The transaction log plays a key role in system recovery. If you do not specify any TRANSACTION LOG clause, or if you omit a path for the file name, it is placed in the same directory as the *.db* file. However, you should place it on a different physical device from the *.db* and *.iq*. It cannot be created on a raw partition.

MIRROR A transaction log mirror is an identical copy of a transaction log, usually maintained on a separate device, for greater protection of your data. By default, Sybase IQ does not use a mirrored transaction log. If you do want to use a transaction log mirror, you must provide a file name. If you use a relative path, the transaction log mirror is created relative to the directory of the Catalog Store (*db-name.db*). Sybase recommends that you always create a mirror copy of the transaction log.

CASE For databases created with CASE RESPECT, all affected values are case sensitive in comparisons and string operations. Database object names such as columns, procedures, or user IDs, are unaffected. Dbspace names are case sensitive for databases created with CASE RESPECT. Password case sensitivity follows data sensitivity unless you specify the PASSWORD CASE clause of CREATE DATABASE.

Note When a database is created with CASE IGNORE, queries might return data in either uppercase or lowercase, depending on the type of index the optimizer chose to use. You can return all uppercase data in such a situation by using this command:

```
SET TEMPORARY OPTION AGGREGATION_PREFERENCE=-2
```

Alternatively, you can use the LOWER or UPPER functions on columns to display the column values in lowercase or uppercase.

This option is provided for compatibility with the ISO/ANSI SQL standard. The default (RESPECT) is that all comparisons are case sensitive. CASE RESPECT provides better performance than CASE IGNORE.

All databases are created with at least one user ID:

```
DBA
```

and password:

```
SQL
```

If you create a database requiring case-sensitive comparisons, the password must be entered in uppercase, unless you specify PASSWORD CASE IGNORE. The user ID is unaffected by the CASE RESPECT setting.

PAGE SIZE The page size for the Anywhere segment of the database (containing the catalog tables) can be 4096, 8192, 16384, or 32768 bytes, with 4096 being the default. Other values for the size are changed to the next larger size. Normally, you should use the default, 4096 (4KB). Large databases might see performance benefits from a page size larger than this default. Smaller values might limit the number of columns your database can support. If you specify a page size smaller than 4096, Sybase IQ uses a page size of 4096.

When you start a database, its page size cannot be larger than the page size of the current server. The server page size is taken from the first set of databases started or is set on the server command line using the -gp command line option.

Command line length for any statement is limited to the Catalog page size. The 4KB default is large enough in most cases; however, in a few cases a larger PAGE SIZE value is needed to accommodate very long commands, such as RESTORE commands that reference numerous dbspaces.

COLLATION The collation sequence used for all string comparisons in the database. The default collation sequence is ISO_BINENG, which provides the best performance. In ISO_BINENG, the collation order is the same as the order of characters in the ASCII character set. All uppercase letters precede all lowercase letters (for example, both ‘A’ and ‘B’ precede ‘a’).

For a list of available collation sequences, see “CP874toUTF8 utility” in Chapter 3, “Database Administration Utilities” in the *Sybase IQ Utility Guide*.

Before creating a database with a nondefault collation, or a custom collating sequence, see Chapter 11, “International Languages and Character Sets” in the *Sybase IQ System Administration Guide*.

ENCRYPTED Encryption makes the data stored in your physical database file unreadable. There are two levels of encryption:

- Simple encryption is equivalent to obfuscation. The data is unreadable, but someone with cryptographic expertise could decipher the data. Simple encryption is achieved by specifying the ENCRYPTED clause with no KEY clause.
- Strong encryption is achieved through the use of a 128-bit algorithm and a security key. The data is unreadable and virtually undecipherable without the key.

Encryption can be specified only during database creation. (To introduce encryption to an existing database requires a complete unload, database recreation, and reload of all data.) To create a strongly encrypted database, specify the ENCRYPTED clause with the KEY clause. As with most passwords, it is best to choose a key value that cannot be easily guessed. We recommend that you choose a value for your key that is at least 16 characters long, contains a mix of uppercase and lowercase, and includes numbers, letters and special characters.

You require this key each time you start the database.

Using the ALGORITHM clause in conjunction with the ENCRYPTED and KEY clauses lets you specify the encryption algorithm. Currently, the only supported algorithm is AES. If the ENCRYPTED clause is used but no algorithm is specified, the default is AES. Encryption is OFF by default.

Warning! Protect your key! Be sure to store a copy of your key in a safe location. A lost key results in a completely inaccessible database, from which there is no recovery.

BLANK PADDING By default, trailing blanks are ignored for comparison purposes (BLANK PADDING ON), and Embedded SQL programs pad strings fetched into character arrays. This option is provided for compatibility with the ISO/ANSI SQL standard.

For example, these two strings are treated as equal in a database created with BLANK PADDING ON:

```
'Smith'
'Smith   '
```

Note CREATE DATABASE no longer supports BLANK PADDING OFF for new databases. This change has no effect on existing databases. You can test the state of existing databases using the BlankPadding database property:

```
select db_property ( 'BlankPadding' )
```

Sybase recommends that you change any existing columns affected by BLANK PADDING OFF, to ensure correct join results. Recreate join columns as CHAR data type, rather than VARCHAR. CHAR columns are always blank padded.

JAVA To use Java in your database, you must install entries for the Sybase runtime Java classes into the catalog system tables. By default, these entries are installed. If you do not need to use Java, you can specify JAVA OFF to avoid installing these entries. Platforms that support JAVA ON include the file "libdbjava7", with a platform-specific suffix, in the /lib directory.

JCONNECT To use the Sybase jConnect for JDBC driver to access system catalog information, you must install jConnect support. Use this option to exclude the jConnect system objects (the default is ON). You can still use JDBC, as long as you do not access system information.

PASSWORD CASE You can specify whether passwords are case sensitive in the database. The case sensitivity of passwords need not be the same as the database's case-sensitivity setting for string comparisons. If you do not specify the case sensitivity of passwords, passwords follow the case sensitivity of the database, which defaults to CASE RESPECT. Extended characters used in passwords (that is, characters above the first 128 in the code page) are case sensitive, regardless of the password case-sensitivity setting.

IQ PATH The path name of the main segment file containing the Sybase IQ data. You can specify an operating system file or a raw partition of an I/O device. (The *Sybase IQ Installation and Configuration Guide* for your platform describes the format for specifying a raw partition.) Sybase IQ automatically detects which type based on the path name you specify. If you use a relative path, the file is created relative to the directory of the Catalog Store (the *.db* file).

See Chapter 8, "Physical Limitations" for an important note about initializing raw devices on Sun Solaris.

IQ SIZE The size in MB of either the raw partition or the operating system file you specify with the IQ PATH clause. For raw partitions, you should always take the default, which allows Sybase IQ to use the entire raw partition; if you specify a value for IQ SIZE, it must match the size of the I/O device or Sybase IQ returns an error. For operating system files, you can specify a value based on the size of your data, from the minimum in Table 6-5 up to a maximum of 128GB. The default for operating system files depends on IQ PAGE SIZE:

Table 6-5: Default and minimum sizes of IQ and Temporary Store files

IQ PAGE SIZE	*IQ SIZE default*	*TEMPORARY SIZE default*	*Minimum explicit IQ SIZE*	*Minimum explicit TEMPORARY SIZE*
65536	4096000	2048000	4MB	2MB
131072	8192000	4096000	8MB	4MB
262144	16384000	8192000	16MB	8MB
524288	32768000	16384000	32MB	16MB

IQ PAGE SIZE The page size in bytes for the Sybase IQ segment of the database (containing the IQ tables and indexes). The value must be a power of 2, from 65536 to 524288 bytes. The default is 131072 (128KB). Other values for the size are changed to the next larger size. The IQ page size determines the default I/O transfer block size and maximum data compression for your database.

For the best performance, Sybase recommends the following minimum IQ page sizes:

- 64KB (IQ PAGE SIZE 65536) for databases whose largest table contains up to 1 billion rows, or a total size less than 8TB. This is the absolute minimum for a new database. On 32-bit platforms, a 64KB IQ page size gives the best performance.
- 128KB (IQ PAGE SIZE 131072) for databases on a 64-bit platform whose largest table contains more than 1 billion rows and fewer than 4 billion rows, or might grow to a total size of 8TB or greater. 128KB is the default IQ page size.
- 256KB (IQ PAGE SIZE 262144) for databases on a 64-bit platform whose largest table contains more than 4 billion rows, or might grow to a total size of 8TB or greater.

Very wide tables, such as tables with multiple columns of wide VARCHAR data (columns from 255 to 32,767 bytes) might need the next larger IQ PAGE SIZE.

BLOCK SIZE he I/O transfer block size in bytes for the Sybase IQ segment of the database. The value must be less than IQ PAGE SIZE, and must be a power of two between 4096 and 32768. Other values for the size are changed to the next larger size. The default value depends on the value of the IQ PAGE SIZE clause. For most applications, this default value is optimum. Before specifying a different value, see Chapter 5, “Managing System Resources” in the *Sybase IQ Performance and Tuning Guide*.

IQ RESERVE Specifies the size in megabytes of space to reserve for the Main IQ Store (IQ_SYSTEM_MAIN dbspace), so that the dbspace can be increased in size in the future. The *sizeMB* parameter can be any number greater than 0. The reserve cannot be changed after the dbspace is created.

When IQ RESERVE is specified, the database uses more space for internal (free list) structures. If reserve size is too large, the space needed for the internal structures can be larger than the specified size, which results in an error.

TEMPORARY RESERVE clause Specifies the size in megabytes of space to reserve for the Temporary IQ Store (IQ_SYSTEM_TEMP dbspace), so that the dbspace can be increased in size in the future. The *sizeMB* parameter can be any number greater than 0. The reserve cannot be changed after the dbspace is created.

When TEMPORARY RESERVE is specified, the database uses more space for internal (free list) structures. If reserve size is too large, the space needed for the internal structures can be larger than the specified size, which results in an error.

Note Reserve and mode for temporary dbspaces are lost if the database is restored from a backup.

MESSAGE PATH The path name of the segment containing the Sybase IQ messages trace file. You must specify an operating system file; the message file cannot be on a raw partition. If you use a relative path or omit the path, the message file is created relative to the directory of the *.db* file.

TEMPORARY PATH The path name of the temporary segment file containing the temporary tables generated by certain queries. You can specify an operating system file or a raw partition of an I/O device. (The *Sybase IQ Installation and Configuration Guide* for your platform describes the format for specifying a raw partition.) Sybase IQ automatically detects which type based on the path name you specify. If you use a relative path or omit the path, the temporary file is created relative to the directory of the *.db* file.

TEMPORARY SIZE The size in MB of either the raw partition or the operating system file you specify with the TEMPORARY PATH clause. For raw partitions, you should always take the default, which allows Sybase IQ to use the entire raw partition. The default for operating system files is always one-half the value of IQ SIZE. If the IQ Store is on a raw partition and the Temporary Store is an operating system file, the default TEMPORARY SIZE is half the size of the IQ Store raw partition.

Side effects

Several operating system files are created.

Standards

- **SQL92** Vendor extension.
- **Sybase** Adaptive Server Enterprise provides a CREATE DATABASE statement, but with different options.

Permissions

The permissions required to execute this statement are set on the server command line, using the -gu option. The default setting is to require DBA authority.

The account under which the server is running must have write permissions on the directories where files are created.

See also

CREATE DBSPACE statement on page 453

DROP DATABASE statement on page 536

CREATE DBSPACE statement

Description

Creates a new dbspace and the associated database file. This file can be on a different device than the initial dbspace.

Syntax

CREATE DBSPACE *dbspace-name* **AS** *filename*
... [{ **IQ STORE** | **IQ TEMPORARY STORE**
| **IQ LOCAL STORE** | **CATALOG STORE** }]
... [[**SIZE**] *file-size*]
... [**RESERVE** *sizeMB*]

Examples

Example 1 On Windows, creates a dbspace called *mydb_tmp_2* to add 200MB to the IQ Temporary Store of the current Sybase IQ database (mydb):

```
CREATE DBSPACE mydb_tmp_2
AS 'e:\\s2\\data\\mydb_2.iqtmp'
IQ TEMPORARY STORE
SIZE 200 ;
```

Example 2 Adds a dbspace on a Windows raw device to a database:

```
CREATE DBSPACE main2 AS '\\\\.\\H:' IQ STORE
```

Always double the backslashes when naming raw devices on Windows in SQL statements.

Usage

CREATE DBSPACE creates a new database file called a dbspace. When a database is first initialized using CREATE DATABASE, it creates several database files by default, including:

- *db-name.db* is the catalog dbspace containing the system tables and stored procedures describing the database and any standard Adaptive Server Anywhere database objects you add. It is known as the Catalog Store, and is named SYSTEM.
- *db-name.iq* is the main data dbspace containing the IQ tables and indexes. It is known as the IQ Store, and is named IQ_SYSTEM_MAIN.
- *db-name.iqtmp* is the initial temporary dbspace containing the temporary tables generated by certain queries. It is known as the IQ Temporary Store, and is called IQ_SYSTEM_TEMP.

CREATE DBSPACE adds a new dbspace to one of these stores. The default is the IQ Store. The dbspace you add can be on a different disk device than the initial dbspace, allowing the creation of stores larger than one physical device.

Note Multiplex databases have a shared IQ Store, where they share all dbspaces, and a local IQ Store. The local IQ Store consists of dbspaces that are managed by only one query server and are not visible to any other query server. To create dbspaces for a multiplex database, see Chapter 5, “Working with Database Objects” in the *Sybase IQ System Administration Guide* for details.

When you create a database or a dbspace, the path for the Temporary Store must be unique. If your CREATE DBSPACE command specifies the identical path and file name for these two stores, you receive an error.

You can create a unique path in any of these ways:

- Specify a different extension for each file (for example, *mydb.iq* and *mydb.iqtmp*)
- Specify a different file name (for example, *mydb.iq* and *mytmp.iq*)
- Specify a different path name (for example, */iqfiles/main/iq* and */iqfiles/temp/iq*) or different raw partitions

---

Warning! On UNIX platforms, to maintain database consistency you must specify file names that are links to different files. Sybase IQ cannot detect the target where linked files point. Even if the file names in the command differ, it is your responsibility to make sure they do not point to the same file.

The *dbspace-name* is an internal name for the dbspace. The *filename* is the actual file name of the dbspace, with a path where necessary. A *filename* without an explicit directory is created in the same directory as the initial dbspace of that store. Any relative directory is relative to that initial dbspace. Each *dbspace-name* must be unique in a database. Dbspace names are case sensitive for databases created with CASE RESPECT.

SIZE clause For operating system files, specifies the size in MB, from 0 to 4194304 (0 to 4 terabytes), of the file you specify in *filename*. See Chapter 8, "Physical Limitations" for platform-specific limits and an important note about initializing raw devices on Sun Solaris. The default depends on the store type and block size. For the IQ Main Store, the default number of bytes equals 1000 * the block size. For the IQ Temporary Store, the default number of bytes equals 100 * the block size. You cannot specify the SIZE clause for the Catalog Store.

A SIZE value of 0 creates a dbspace of minimum size, which is 1000 blocks for IQ Main Store and 100 blocks for IQ Temporary Store.

For raw partitions, do not specify SIZE explicitly. Sybase IQ sets this parameter to the maximum raw partition size automatically, and returns an error if you attempt to specify another size.

RESERVE clause Specifies the size in megabytes of space to reserve, so that the dbspace can be increased in size in the future. The *sizeMB* parameter can be any number greater than 0. The reserve cannot be changed after the dbspace is created.

When RESERVE is specified, the database uses more space for internal (free list) structures. If reserve size is too large, the space needed for the internal structures can be larger than the specified size, which results in an error.

Note Reserve and mode for temporary dbspaces are lost if the database is restored from a backup.

A database can have up to 2047 dbspaces, including the initial dbspaces created when you create the database. However, your operating system might limit the number of files per database.

Side effects

Automatic commit. Automatic checkpoint.

Standards

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

Permissions
Must have DBA authority.

See also
DROP statement on page 533

Chapter 5, "Working with Database Objects," in the *Sybase IQ System Administration Guide.*

CREATE DOMAIN statement

Description
Creates a user-defined data type in the database.

Syntax
CREATE { **DOMAIN** | **DATATYPE** } *domain-name data-type*
... [[**NOT**] **NULL**]
... [**DEFAULT** *default-value*]

Parameters
domain-name:
identifier

data-type:
built-in data type, with precision and scale

default-value:
special-value
| *string*
| *global variable*
| [-] *number*
| (*constant-expression*)
| *built-in-function*(*constant-expression*)
| AUTOINCREMENT
| CURRENT DATABASE
| CURRENT REMOTE USER
| NULL
| TIMESTAMP
| LAST USER

special-value:
CURRENT { DATE | TIME | TIMESTAMP | USER | PUBLISHER }
| USER

Examples
The following statement creates a data type named address, which holds a 35-character string, and which may be NULL:

```
CREATE DOMAIN address CHAR( 35 ) NULL
```

Usage

User-defined data types are aliases for built-in data types, including precision and scale values, where applicable. They improve convenience and encourage consistency in the database.

Sybase recommends that you use CREATE DOMAIN, rather than CREATE DATATYPE, as CREATE DOMAIN is the ANSI/ISO SQL3 term.

The user who creates a data type is automatically made the owner of that data type. No owner can be specified in the CREATE DATATYPE statement. The user-defined data type name must be unique, and all users can access the data type without using the owner as prefix.

User-defined data types are objects within the database. Their names must conform to the rules for identifiers. User-defined data type names are always case insensitive, as are built-in data type names.

By default, user-defined data types allow NULLs unless the allow_nulls_by_default option is set to OFF. In this case, new user-defined data types by default do not allow NULLs. The nullability of a column created on a user-defined data type depends on the setting of the definition of the user-defined data type, not on the setting of the allow_nulls_by_default option when the column is referenced. Any explicit setting of NULL or NOT NULL in the column definition overrides the user-defined data type setting.

The CREATE DOMAIN statement allows you to specify DEFAULT values on user-defined data types. The DEFAULT value specification is inherited by any column defined on the data type. Any DEFAULT value explicitly specified on the column overrides that specified for the data type. For more information on the use of column DEFAULT values, see “Using column defaults” in Chapter 9, “Ensuring Data Integrity” in the *Sybase IQ System Administration Guide*.

The CREATE DOMAIN statement lets you incorporate a rule, called a CHECK condition, into the definition of a user-defined data type.

Sybase IQ enforces CHECK constraints for base, global temporary. local temporary tables, and user-defined data types.

To drop the data type from the database, use the DROP statement. You must be either the owner of the data type or have DBA authority in order to drop a user-defined data type.

Side effects

Automatic commit.

Standards

- **SQL92** Intermediate-level feature.

- **Sybase** Not supported by Adaptive Server Enterprise. Transact-SQL provides similar functionality using the sp_addtype system procedure and the CREATE DEFAULT and CREATE RULE statements.

Permissions

Must have RESOURCE authority.

See also

DROP statement on page 533

Chapter 4, “SQL Data Types”

CREATE EVENT statement

Description

Defines an event and its associated handler for automating predefined actions. Also defines scheduled actions.

Syntax

```
CREATE EVENT event-name
... [ TYPE event-type
        [ WHERE trigger-condition [ AND trigger-condition ], ... ]
    | SCHEDULE schedule-spec, ... ]
... [ ENABLE | DISABLE ]
... [ AT { CONSOLIDATED | REMOTE | ALL } ]
...[ HANDLER
     BEGIN
...
     END ]
```

Parameters

event-type:

BackupEnd | "Connect"
| ConnectFailed | DatabaseStart
| DBDiskSpace | "Disconnect"
| GlobalAutoincrement | GrowDB
| GrowLog | GrowTemp
| LogDiskSpace | "RAISERROR"
| ServerIdle | TempDiskSpace

trigger-condition:

event_condition(*condition-name*) { = | < | > | != | <= | >= } *value*

schedule-spec:

[*schedule-name*]
{ START TIME *start-time* | BETWEEN *start-time* AND *end-time* }
[EVERY *period* { HOURS | MINUTES | SECONDS }]
[ON { (*day-of-week, ...*) | (*day-of-month, ...*) }]
[START DATE *start-date*]

event-name | *schedule-name*:
identifier

day-of-week :
string

day-of-month | *value* | *period* :
integer

start-time | *end-time* :
time

start-date :
date

Examples

- This example instructs the database server to carry out an automatic incremental backup daily at 1 a.m.:

```
CREATE EVENT IncrementalBackup
SCHEDULE
START TIME '1:00AM' EVERY 24 HOURS
HANDLER
BEGIN
   BACKUP DATABASE INCREMENTAL
   TO 'backups/daily.incr'
END
```

- This example instructs the database server to call the system stored procedure sp_iqspaceused every 10 minutes, then store in a table the returned current date and time, the current number of connections to the database, and current information about the use of Main and Temporary IQ Store:

```
CREATE TABLE mysummary(dt DATETIME,
   users INT, mainKB UNSIGNED BIGINT,
   mainPC UNSIGNED INT,
   tempKB UNSIGNED BIGINT,
   tempPC UNSIGNED INT) ;

CREATE EVENT mysummary
   SCHEDULE sched_mysummary
      START TIME '00:01 AM' EVERY 10 MINUTES
   HANDLER
   BEGIN
      DECLARE mt UNSIGNED BIGINT;
      DECLARE mu UNSIGNED BIGINT;
      DECLARE tt UNSIGNED BIGINT;
      DECLARE tu UNSIGNED BIGINT;
      DECLARE conncount UNSIGNED INT;
```

```
        SET connCount = DB_PROPERTY('ConnCount');
        CALL SP_IQSPACEUSED(mt,mu,tt,tu);

        INSERT INTO mysummary VALUES( NOW(),
        connCount, mu, (mu*100)/mt, tu,
        (tu*100)/tt );

    END ;
```

For more examples, see "Defining trigger conditions for events" in Chapter 18, "Automating Tasks Using Schedules and Events" in the *Sybase IQ System Administration Guide*.

Usage

Events can be used in two main ways:

- **Scheduling actions** The database server carries out a set of actions on a schedule of times. You can use this capability to schedule backups, validity checks, queries to fill up reporting tables, and so on.
- **Event handling actions** The database server carries out a set of actions when a predefined event occurs. The events that can be handled include disk space restrictions (when a disk fills beyond a specified percentage), when the server is idle, and so on.

An event definition includes two distinct pieces. The **trigger condition** can be an occurrence, such as a disk filling up beyond a defined threshold. A **schedule** is a set of times, each of which acts as a trigger condition. When a trigger condition is satisfied, the **event handler** executes. The event handler includes one or more actions specified inside a compound statement (BEGIN... END).

If no trigger condition or schedule specification is supplied, only an explicit TRIGGER EVENT statement can trigger the event. During development, you might wantmight to develop and test event handlers using TRIGGER EVENT, and add the schedule or WHERE clause once testing is complete.

Event errors are logged to the database server console.

When event handlers are triggered, the server makes context information, such as the connection ID that caused the event to be triggered, available to the event handler using the EVENT_PARAMETER function.

Note Although statements that return result sets are disallowed in events, you can allow an event to call a stored procedure and insert the procedure results into a temporary table. For details, see "Extraction and events" in Chapter 7, "Moving Data In and Out of Databases,"*Sybase IQ System Administration Guide*.

CREATE EVENT The event name is an identifier. An event has a creator, which is the user creating the event, and the event handler executes with the permissions of that creator. This is the same as stored procedure execution. You cannot create events owned by other users.

You can list event names by querying the system table SYSEVENT. For example:

```
SELECT event_id, event_name FROM SYS.SYSEVENT
```

TYPE The *event-type* is one of the listed set of system-defined event types. The event types are case insensitive. To specify the conditions under which this *event-type* triggers the event, use the WHERE clause.

- **DiskSpace event types** If the database contains an event handler for one of the DiskSpace types, the database server checks the available space on each device associated with the relevant file every 30 seconds.

  In the event the database has more than one dbspace, on separate drives, DBDiskSpace checks each drive and acts depending on the lowest available space.

  The LogDiskSpace event type checks the location of the transaction log and any mirrored transaction log, and reports based on the least available space.

  The disk space event types require Windows and are not available on UNIX platforms.

- **Globalautoincrement event type** This event fires when the GLOBAL AUTOINCREMENT default value for a table is within one percent of the end of its range. A typical action for the handler could be to request a new value for the GLOBAL_DATABASE_ID option.

  You can use the EVENT_CONDITION function with RemainingValues as an argument for this event type.

- **ServerIdle event type** If the database contains an event handler for the ServerIdle type, the server checks for server activity every 30 seconds.

WHERE clause The trigger condition determines the condition under which an event is fired. For example, to take an action when the disk containing the transaction log becomes more than 80% full, use the following triggering condition:

```
...
WHERE event_condition( 'LogDiskSpacePercentFree' ) < 20
...
```

The argument to the EVENT_CONDITION function must be valid for the event type.

You can use multiple AND conditions to make up the WHERE clause, but you cannot use OR conditions or other conditions.

For information on valid arguments, see "EVENT_CONDITION function [System]" on page 300.

SCHEDULE This clause specifies when scheduled actions are to take place. The sequence of times acts as a set of triggering conditions for the associated actions defined in the event handler.

You can create more than one schedule for a given event and its associated handler. This permits complex schedules to be implemented. While it is compulsory to provide a schedule name when there is more than one schedule, it is optional if you provide only a single schedule.

You can list schedule names by querying the system table SYSSCHEDULE. For example:

```
SELECT event_id, sched_name FROM SYS.SYSSCHEDULE
```

Each event has a unique event ID. Use the event_id columns of SYSEVENT and SYSSCHEDULE to match the event to the associated schedule.

When a nonrecurring scheduled event has passed, its schedule is deleted, but the event handler is not deleted.

Scheduled event times are calculated when the schedules are created, and again when the event handler completes execution. The next event time is computed by inspecting the schedule or schedules for the event, and finding the next schedule time that is in the future. If an event handler is instructed to run every hour between 9:00 and 5:00, and it takes 65 minutes to execute, it runs at 9:00, 11:00, 1:00, 3:00, and 5:00. If you want execution to overlap, you must create more than one event.

The subclauses of a schedule definition are as follows:

- **START TIME** The first scheduled time for each day on which the event is scheduled. If a START DATE is specified, the START TIME refers to that date. If no START DATE is specified, the START TIME is on the current day (unless the time has passed) and each subsequent day.
- **BETWEEN ... AND** A range of times during the day outside of which no scheduled times occur. If a START DATE is specified, the scheduled times do not occur until that date.
- **EVERY** An interval between successive scheduled events. Scheduled events occur only after the START TIME for the day, or in the range specified by BETWEEN ... AND.
- **ON** A list of days on which the scheduled events occur. The default is every day. These can be specified as days of the week or days of the month.

  Days of the week are Monday, Tuesday, and so on. The abbreviated forms of the day, such as Mon, Tue, and so on, may also be used. The database server recognizes both full-length and abbreviated day names in any of the languages supported by Sybase IQ.

  Days of the month are integers from 0 to 31. A value of 0 represents the last day of any month.
- **START DATE** The date on which scheduled events are to start occurring. The default is the current date.

Each time a scheduled event handler is completed, the next scheduled time and date is calculated.

1 If the EVERY clause is used, find whether the next scheduled time falls on the current day, and is before the end of the BETWEEN ... AND range. If so, that is the next scheduled time.

2 If the next scheduled time does not fall on the current day, find the next date on which the event is to be executed.

3 Find the START TIME for that date, or the beginning of the BETWEEN ... AND range.

ENABLE | *DISABLE* By default, event handlers are enabled. When DISABLE is specified, the event handler does not execute even when the scheduled time or triggering condition occurs. A TRIGGER EVENT statement does *not* cause a disabled event handler to be executed.

AT To execute events at remote or consolidated databases in a SQL Remote setup, use this clause to restrict the databases at which the event is handled. By default, all databases execute the event.

HANDLER Each event has one handler. Like the body of a stored procedure, the handler is a compound statement. There are some differences, though: you can use an EXCEPTION clause within the compound statement to handle errors, but not the ON EXCEPTION RESUME clause provided within stored procedures.

Side effects

Automatic commit.

The actions of an event handler are committed if no error is detected during execution, and rolled back if errors are detected.

Standards

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

Permissions

Must have DBA authority.

Event handlers execute on a separate connection, with the permissions of the event owner. To execute with permissions other than DBA, you can call a procedure from within the event handler: the procedure executes with the permissions of its owner. The separate connection does not count towards the ten-connection limit of the personal database server.

See also

ALTER EVENT statement on page 401

BEGIN... END statement on page 422

COMMENT statement on page 435

DROP statement on page 533

TRIGGER EVENT statement on page 658

Chapter 18, “Automating Tasks Using Schedules and Events,” in the *Sybase IQ System Administration Guide.*

CREATE EXISTING TABLE statement

Description

Creates a new proxy table representing an existing object on a remote server.

Syntax

CREATE EXISTING TABLE [*owner*.]*table_name*
[(*column-definition*, ...)]
AT *'location-string'*

Parameters

column-definition:
column-name data-type [NOT NULL]

location-string:
remote-server-name.[*db-name*].[*owner*].*object-name*
| *remote-server-name*;[*db-name*];[*owner*];*object-name*

Examples

- This example creates a proxy table named blurbs for the blurbs table at the remote server server_a:

```
CREATE EXISTING TABLE blurbs
( author_id id not null,
copy text not null)
AT 'server_a.db1.joe.blurbs'
```

- This example creates a proxy table named blurbs for the blurbs table at the remote server server_a. Sybase IQ derives the column list from the metadata it obtains from the remote table:

```
CREATE EXISTING TABLE blurbs
AT 'server_a.db1.joe.blurbs'
```

- This example creates a proxy table named rda_employee for the employee table at the Sybase IQ remote server asiqdemo:

```
CREATE EXISTING TABLE rda_employee
AT 'asiqdemo..dba.employee'
```

Usage

CREATE EXISTING TABLE is a variant of the CREATE TABLE statement. The EXISTING keyword is used with CREATE TABLE to specify that a table already exists remotely and that its metadata is to be imported into Sybase IQ. This establishes the remote table as a visible entity to its users. Sybase IQ verifies that the table exists at the external location before it creates the table.

Tables used as proxy tables cannot have names longer than 30 characters.

If the object does not exist (either host data file or remote server object), the statement is rejected with an error message.

Index information from the host data file or remote server table is extracted and used to create rows for the system table sysindexes. This defines indexes and keys in server terms and enables the query optimizer to consider any indexes that might exist on this table.

Referential constraints are passed to the remote location when appropriate.

If column definitions are not specified, Sybase IQ derives the column list from the metadata it obtains from the remote table. If column definitions are specified, Sybase IQ verifies the column definitions. Column names, data types, lengths, and null properties are checked for the following:

- Column names must match identically (although case is ignored).
- Data types in CREATE EXISTING TABLE must match or be convertible to the data types of the column on the remote location. For example, a local column data type is defined as NUMERIC, whereas the remote column data type is MONEY.
- Each column's NULL property is checked. If the local column's NULL property is not identical to the remote column's NULL property, a warning message is issued, but the statement is not aborted.
- Each column's length is checked. If the lengths of CHAR, VARCHAR, BINARY, DECIMAL, and NUMERIC columns do not match, a warning message is issued, but the command is not aborted. You might choose to include only a subset of the actual remote column list in your CREATE EXISTING statement.
- AT specifies the location of the remote object. The AT clause supports the semicolon (;) as a delimiter. If a semicolon is present anywhere in the location string, the semicolon is the field delimiter. If no semicolon is present, a period is the field delimiter. This allows file names and extensions to be used in the database and owner fields. Semicolon field delimiters are used primarily with server classes not currently supported; however, you can also use them where a period would also work as a field delimiter. For example, the following statement maps the table proxy_a to the Adaptive Server Anywhere database mydb on the remote server myasa:

  ```
  CREATE EXISTING TABLE
  proxy_a1
  AT 'myasa;mydb;;a1'
  ```

Side effects

Automatic commit.

Standards

- **SQL92** Entry-level feature.

- **Sybase** Supported by Open Client/Open Server.

Permissions

Must have RESOURCE authority. To create a table for another user, you must have DBA authority.

See also

CREATE TABLE statement on page 499

Chapter 17, "Server Classes for Remote Data Access," and Chapter 16, "Accessing Remote Data," in the *Sybase IQ System Administration Guide*.

CREATE EXTERNLOGIN statement

Description

Assigns an alternate login name and password to be used when communicating with a remote server.

Syntax

CREATE EXTERNLOGIN *login-name*
TO *remote-server*
REMOTE LOGIN *remote-user*
[**IDENTIFIED BY** *remote-password*]

Examples

Maps the local user named DBA to the user sa with password 4TKNOX when connecting to the server sybase1:

```
CREATE EXTERNLOGIN dba
TO sybase1
REMOTE LOGIN sa
IDENTIFIED BY 4TKNOX
```

Usage

By default, Sybase IQ uses the names and passwords of its clients whenever it connects to a remote server on behalf of those clients. CREATE EXTERNLOGIN assigns an alternate login name and password to be used when communicating with a remote server. It stores the password internally in encrypted form. The *remote_server* must be known to the local server by an entry in the sysservers table. For more information, see CREATE SERVER statement on page 494.

Sites with automatic password expiration should plan for periodic updates of passwords for external logins.

CREATE EXTERNLOGIN cannot be used from within a transaction.

login-name Specifies the local user login name. When using integrated logins, the *login-name* is the database user to which the Windows user ID is mapped.

TO The TO clause specifies the name of the remote server.

REMOTE LOGIN The REMOTE LOGIN clause specifies the user account on *remote-server* for the local user *login-name*.

IDENTIFIED BY The IDENTIFIED BY clause specifies remote-password is the password for remote-user

The *remote-user* and *remote-password* combination must be valid on *remote-server*.

Side effects

Automatic commit.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions

Only the *login-name* and the DBA account can add or modify an external login for *login-name*.

See also

DROP EXTERNLOGIN statement on page 538

CREATE FUNCTION statement

Description

Creates a new function in the database.

Syntax

CREATE FUNCTION [*owner*.]*function-name* **(** [*parameter*, …] **)**
RETURNS *data-type routine-characteristics*
{ *compound-statement*
| **AS** *tsql-compound-statement*
| *external-name* }

Parameters

parameter:
IN *parameter-name data-type*

routine-characteristics:
ON EXCEPTION RESUME | [NOT] DETERMINISTIC

tsql-compound-statement:
sql-statement
sql-statement ...

external-name:
EXTERNAL NAME *library-call*
| EXTERNAL NAME *java-call* LANGUAGE JAVA

library-call:
'[*operating-system*:]*function-name@library.dll*; ...'

operating-system:
WindowsNT | UNIX

java-call:
'[*package-name*.]*class-name*.*method-name method-signature*'

method-signature:
([*field-descriptor*, ...]) *return-descriptor*

field-descriptor | *return-descriptor*:
Z | B | S | I | J | F | D | C | V | [*descriptor* | L*class-name*;

Examples

Example 1 Concatenates a firstname string and a lastname string:

```
CREATE FUNCTION fullname (
    firstname CHAR(30),
    lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
    DECLARE name CHAR(61);
    SET name = firstname || ' ' || lastname;
    RETURN (name);
END
```

The following examples illustrate the use of the fullname function.

- To return a full name from two supplied strings, enter:

```
SELECT fullname ('joe','smith')
```

fullname('joe', 'smith')
joe smith

- To list the names of all employees, enter:

```
SELECT fullname (emp_fname, emp_lname)
FROM employee
```

fullname (emp_fname, emp_lname)
Fran Whitney
Matthew Cobb
Philip Chin
Julie Jordan
Robert Breault
...

Example 2 Uses Transact-SQL syntax:

```
CREATE FUNCTION DoubleIt ( @Input INT )
```

```
RETURNS INT
AS
DECLARE @Result INT
SELECT @Result = @Input * 2
RETURN @Result
```

The statement `SELECT DoubleIt( 5 )` returns a value of `10`.

Example 3 Creates an external function written in Java:

```
CREATE FUNCTION dba.encrypt( IN name char(254) )
RETURNS VARCHAR
EXTERNAL NAME
'Scramble.encrypt (Ljava/lang/String;)Ljava/lang/
String;'
LANGUAGE JAVA
```

Usage

CREATE FUNCTION creates a user-defined function in the database. A function can be created for another user by specifying an owner name. Subject to permissions, a user-defined function can be used in exactly the same way as other nonaggregate functions.

The following describes each clause of the CREATE FUNCTION statement.

CREATE FUNCTION Parameter names must conform to the rules for database identifiers. They must have a valid SQL data type, and must be prefixed by IN, signifying that the argument is an expression that provides a value to the function.

compound-statement A set of SQL statements bracketed by BEGIN and END, and separated by semicolons. See BEGIN... END statement on page 422

tsql-compound-statement A batch of Transact-SQL statements. See "Transact-SQL batch overview" on page 814, and CREATE PROCEDURE statement [T-SQL] on page 491.

EXTERNAL NAME A function using the EXTERNAL NAME clause is a wrapper around a call to a function in an external library. A function using EXTERNAL NAME can have no other clauses following the RETURNS clause. The library name may include the file extension, which is typically .dll on Windows, and .so on UNIX. In the absence of the extension, the software appends the platform-specific default file extension for libraries.

For information about external library calls, see "Calling external libraries from procedures" in Chapter 8, "Using Procedures and Batches" in the *Sybase IQ System Administration Guide*.

EXTERNAL NAME LANGUAGE JAVA A function that uses EXTERNAL NAME with a LANGUAGE JAVA clause is a wrapper around a Java method.

For information on calling Java procedures, see CREATE PROCEDURE statement on page 485.

ON EXCEPTION RESUME Use Transact-SQL -like error handling. For more information, see CREATE PROCEDURE statement on page 485.

NOT DETERMINISTIC A function specified as NOT DETERMINISTIC is re-evaluated each time it is called in a query. The results of functions not specified in this manner may be cached for better performance, and re-used each time the function is called with the same parameters during query evaluation.

Functions that have side effects such as modifying the underlying data should be declared as NOT DETERMINISTIC. For example, a function that generates primary key values and is used in an INSERT ... SELECT statement should be declared NOT DETERMINISTIC:

```
CREATE FUNCTION keygen( increment INTEGER )
RETURNS INTEGER
NOT DETERMINISTIC
BEGIN
  DECLARE keyval INTEGER;
  UPDATE counter SET x = x + increment;
  SELECT counter.x INTO keyval FROM counter;
  RETURN keyval
END
INSERT INTO new_table
SELECT keygen(1), ...
FROM old_table
```

Functions may be declared as DETERMINISTIC if they always return the same value for given input parameters.

Unless they are declared NOT DETERMINISTIC, all user-defined functions return a consistent result for the same parameters and are free of side effects. That is, the server assumes that two successive calls to the same function with the same parameters will return the same result, and will not have any unwanted side effects on the query's semantics.

Note User-defined functions are processed by Adaptive Server 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 certain cases, differences in semantics between ASA 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 Anywhere treats CHAR data as if it were VARCHAR.

To modify a user-defined function, or to hide the contents of a function by scrambling its definition, use the ALTER FUNCTION statement. For more information, see the *Adaptive Server Anywhere SQL Reference*.

Side effects

Automatic commit.

Standards

- **SQL/92** Persistent Stored Module feature.
- **Sybase** Not supported by Adaptive Server Enterprise.

Permissions

Must have RESOURCE authority.

External functions, including Java functions, must have DBA authority.

See also

BEGIN... END statement on page 422

CREATE PROCEDURE statement on page 485

DROP statement on page 533

RETURN statement on page 627

Chapter 8, “Using Procedures and Batches,” in the *Sybase IQ System Administration Guide*

“ALTER FUNCTION statement” in the *Adaptive Server Anywhere SQL Reference*.

CREATE INDEX statement

Description

Creates an index on a specified table, or pair of tables.

Syntax

CREATE [**UNIQUE**] [*index-type*] **INDEX** *index-name*
... **ON** [*owner*.]*table-name*
... (*column-name* [, *column-name*] ...)
... [{ **IN** | **ON** } *dbspace-name*]
... [**NOTIFY** *integer*]
... [**DELIMITED BY** '*separators-string* ']
... [**LIMIT** *maxwordsize-integer*]

Parameters

index-type:
{ CMP | HG | HNG | LF | WD | DATE | TIME | DTTM }

Examples

- Creates a Compare index on the projected_earnings and current_earnings columns. These columns are decimal columns with identical precision and scale.

```
CREATE
CMP INDEX proj_curr_cmp
ON sales_data
( projected_earnings, current_earnings )
```

- Creates a High_Group index on the sales_order_items table for the product ID column:

```
CREATE HG INDEX item_prod_hg
ON sales_order_items
( prod_id )
```

- Creates a Low_Fast index on the sales_order_items table for the same product ID column without any notification messages:

```
CREATE LF INDEX item_prod
ON sales_order_items
( prod_id )
 NOTIFY 0
```

- Creates a WD index on the earnings_report table. Specify that the delimiters of strings are space, colon, semicolon, and period. Limit the length of the strings to 25.

```
CREATE WD INDEX earnings_wd
ON earnings_report_table(varchar)
DELIMITED BY ' :;.'
LIMIT 25
```

- Create a DTTM index on the sales_order table for the order_date column.

```
CREATE DTTM INDEX order_dttm
ON sales_order
( order_date )
```

Usage

The CREATE INDEX statement creates an index on the specified column of the named table. Once an index is created, it is never referenced in a SQL statement again except to delete it using the DROP INDEX statement.

For columns in Sybase IQ tables, you can specify an *index-type* of HG (High_Group), HNG (High_Non_Group), LF (Low_Fast), WD (Word), DATE, TIME, or DTTM (Datetime). If you do not specify an *index-type*, an HG index is created by default.

To create an index on the relationship between two columns in an IQ table, you can specify an *index-type* of CMP (Compare). Columns must be of identical data type, precision and scale. For a CHAR, VARCHAR, BINARY or VARBINARY column, precision means that both columns have the same width.

For maximum query speed, the correct type of index for a column depends on:

- The number of unique values in the column
- How the column is going to be used in queries
- The amount of disk space available

The *Sybase IQ System Administration Guide* describes the index types in detail and tells how to determine the appropriate index types for your data.

You can specify multiple indexes on a column of an IQ table, but these must be of different index types. CREATE INDEX does not let you add a duplicate index type. Sybase IQ chooses the fastest index available for the current query or portion of the query. However, each additional index type might significantly add to the space requirements of that table.

column-name Specifies the name of the column to be indexed. 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 on page 553.) If a column name has characters other than letters, digits, and underscore, enclose it in quotation marks (“”).

When you omit UNIQUE, you can specify only an HG index. Foreign keys require nonunique HG indexes and composite foreign keys require nonunique composite HG indexes. The multicolumn composite key for both unique and nonunique HG indexes has a maximum width of 5300 bytes. CHAR or VARCHAR data cannot be more than 255 bytes when it is part of a composite key or single-column HG, LF, HNG, DATE, TIME, or DTTM indexes.

UNIQUE UNIQUE ensures that no two rows in the table have identical values in all the columns in the index. Each index key must be unique or contain a NULL in at least one column. You can create unique HG indexes with more than one column, but you cannot create multicolumn indexes using other index types. You cannot specify UNIQUE with the CMP, HNG, WD, DATE, TIME, or DTTM index types.

Sybase IQ allows the use of NULL in data values on a user created unique multicolumn HG index, if the column definition allows for NULL values and a constraint (primary key or unique) is not being enforced. See "Multicolumn indexes" in "Notes" on page 477 for more information.

Index placement An index is always placed in the same type of dbspace (IQ Store or Temporary Store) as its table. When you load the index, the data is spread across any database files of that type with room available. Although the CREATE INDEX command lets you specify the *dbspace-name* IQ_SYSTEM_TEMP or IQ_SYSTEM_MAIN, this option has no real effect for Sybase IQ indexes. Sybase IQ ensures that any *dbspace-name* you specify is appropriate for the index. If you try to specify IQ_SYSTEM_MAIN for indexes on temporary tables, or vice versa, you receive an error. Dbspace names are case sensitive for databases created with CASE RESPECT.

DELIMITED BY Specifies separators to use in parsing a column string into the words to be stored in that column's WD index. If you omit this clause or specify the value as an empty string, Sybase IQ uses the default set of separators. The default set of separators is designed for the default collation order (ISO-BINENG). It includes all 7-bit ASCII characters that are not 7-bit ASCII alphanumeric characters, except for the hyphen and the single quotation mark. The hyphen and the single quotation mark are part of words by default. There are 64 separators in the default separator set. For example, if the column value is this string:

```
The cat is on the mat
```

and the database was created with the CASE IGNORE setting using default separators, the following words are stored in the WD index from this string:

```
cat is mat on the
```

If you specify multiple DELIMITED BY and LIMIT clauses, no error is returned, but only the last clause of each type is used.

separators-string The separators string must be a sequence of 0 or more characters in the collation order used when the database was created. Each character in the separators string is treated as a separator. If there are no characters in the separators string, the default set of separators is used. (Each separator must be a single character in the collation sequence being used.) There cannot be more than 256 characters (separators) in the separators string.

To specify tab as a delimiter, you can either type a <TAB> character within the separator string, or use the hexadecimal ASCII code of the tab character, \x09. "\t" specifies two separators, \ and the letter t. To specify newline as a delimiter, you can type a <RETURN> character or the hexadecimal ASCII code \x0a.

For example, the clause `DELIMITED BY ' :;.\/t'` specifies these seven separators: space : ; . \ / t

Table 6-6: Tab and newline as delimiters

For these delimiters	Use this separators string in the DELIMITED BY clause
tab	`'  '` (type <TAB>) or `'\x09'`
newline	`'  '` (type <RETURN>) or `'\x0a'`

LIMIT Can be used for the creation of the WD index only. Specifies the maximum word length that is permitted in the WD index. Longer words found during parsing causes an error. The default is 255 bytes. The minimum permitted value is 1 and the maximum permitted value is 255. If the maximum word length specified in the CREATE INDEX statement or determined by default exceeds the column width, the used maximum word length is silently reduced to the column width. Using a lower maximum permitted word length allows insertions, deletions, and updates to use less space and time. The empty word (two adjacent separators) is silently ignored. After a WD index is created, any insertions into its column are parsed using the separators and maximum word size determined at create time. These separators and maximum word size cannot be changed after the index is created.

NOTIFY Gives notification messages after *n* records are successfully added for the index. The messages are sent to the standard output device. A message contains information about memory usage, database space, and how many buffers are in use. The default is 100,000 records. To turn off NOTIFY, set it to 0.

Notes

- **Index ownership** There is no way to specify the index owner in the CREATE INDEX statement. Indexes are automatically owned by the owner of the table on which they are defined. The index name must be unique for each owner.
- **No indexes on views** Indexes cannot be created for views.
- **Index name** The name of each index must be unique for a given table.
- **Exclusive table use** CREATE INDEX is prevented whenever the statement affects a table currently being modified by another connection. However, queries are allowed on a table that is also adding an index.
- **CHAR columns** After a WD index is created, any insertions into its column are parsed using the separators, and maximum word size cannot be changed after the index is created.

  For CHAR columns, Sybase recommends that you specify a space as at least one of the separators or use the default separator set. Sybase IQ automatically pads CHAR columns to the maximum column width. If your column contains blanks in addition to the character data, queries on WD indexed data might return misleading results. For example, column company_name contains two words delimited by a separator, but the second word is blank padded:

  ```
  'Concord' 'Farms                    '
  ```

  Suppose that a user entered the following query:

  ```
  SELECT COUNT(*)FROM customers WHERE company_name
  contains ('Farms')
  ```

  The parser determines that the string contains:

  ```
  'Farms                              '
  ```

  instead of:

  ```
  'Farms'
  ```

  and returns 0 instead of 1. You can avoid this problem by using VARCHAR instead of CHAR columns.

- **Data types** You cannot use CREATE INDEX to create an index on a column with BIT data. Only the default index, CMP index, or WD index can be created on CHAR and VARCHAR data with more than 255 bytes. Only the default index and CMP index can be created on VARBINARY data with more than 255 bytes. In addition, you cannot create an HNG index or a CMP index on a column with FLOAT, REAL, or DOUBLE data. A TIME index can be created only on a column having the data type TIME. A DATE index can be created only on a column having the data type DATE. A DTTM index can be created only on a column having the data type DATETIME or TIMESTAMP.
- **Multicolumn indexes** You can create a unique or nonunique HG index with more than one column. (Sybase IQ implicitly creates a nonunique HG index on a set of columns that make up a foreign key.) HG and CMP are the only types of indexes that can have multiple columns. You cannot create a unique HNG or LF index with more than one column. You cannot create a DATE, TIME, or DTTM index with more than one column.

  The maximum width of a multicolumn concatenated key is 5KB (5300 bytes). The number of columns allowed depends on how many columns can fit into 1KB. CHAR or VARCHAR data greater than 255 bytes is not allowed as part of a composite key in single-column HG, LF, HNG, DATE, TIME, or DTTM indexes.

  Multicolumn indexes on base tables are *not* replicated in join indexes created using those base tables.

  An INSERT on a multicolumn index must include all columns of the index.

  Sybase IQ allows the use of NULL in data values on a user created unique multicolumn HG index, if the column definition allows for NULL values and a constraint (primary key or unique) is not being enforced. The rules for this feature are as follows:

  - A NULL is treated as an undefined value.
  - Multiple rows with NULL value(s) in a unique index column(s) are allowed.

    1 In a single column index, multiple rows with a NULL value in an index column are allowed.

    2 In a multicolumn index, multiple rows with a NULL value in index column(s) are allowed, as long as non-NULL values in the rest of the columns guarantee uniqueness in that index.

    3 In a multicolumn index, multiple rows with NULL values in all columns participating in the index are allowed.

The following examples illustrate these rules. Given the table table1:

```
CREATE TABLE table1
(c1 INT NULL, c2 INT NULL, c3 INT NOT NULL);
```

Create a unique single column HG index on a column that allows NULLs:

```
CREATE UNIQUE HG INDEX c1_hg1 ON table1 (c1);
```

According to rule 1 above, you can insert a NULL value into an index column in multiple rows:

```
INSERT INTO table1(c1,c2,c3) VALUES (NULL,1,1);
INSERT INTO table1(c1,c2,c3) VALUES (NULL,2,2);
```

Create a unique multicolumn HG index on a columns that allows NULLs:

```
CREATE UNIQUE HG INDEX c1c2_hg2 ON table1(c1,c2);
```

According to rule 2 above, you must guarantee uniqueness in the index. The following INSERT does not succeed, since the multicolumn index c1c2_hg2 on row 1 and row 3 has the same value:

```
INSERT INTO table1(c1,c2,c3) VALUES (NULL,1,3);
```

The following INSERT operations are successful, however, according to rules 1 and 3:

```
INSERT INTO table1(c1,c2,c3) VALUES (NULL,NULL,3);
INSERT INTO table1(c1,c2,c3) VALUES (NULL,NULL,4);
```

Uniqueness is preserved in the multicolumn index.

The following UPDATE operation is successful, as rule 3 allows multiple rows with NULL values in all columns in the multicolumn index:

```
UPDATE table1 SET c2=NULL WHERE c3=1
```

When a multicolumn HG index is governed by a unique constraint, a NULL value is not allowed in any column participating in the index.

- **Parallel index creation** You can use the BEGIN PARALLEL IQ ... END PARALLEL IQ statement to group CREATE INDEX statements on multiple IQ tables, so that they execute as though they are a single DDL statement. See BEGIN PARALLEL IQ ... END PARALLEL IQ statement on page 425 for more information.

Warning! Using the CREATE INDEX command on a local temporary table containing uncommitted data fails and generates the following error message: "Local temporary table, <*tablename*>, must be committed in order to create an index." Commit the data in the local temporary table before creating an index.

Side effects

Automatic commit.

Standards

- **SQL92** Vendor extension.
- **Sybase** Adaptive Server Enterprise has a more complex CREATE INDEX statement than Sybase IQ. While the Adaptive Server Enterprise syntax is permitted in Sybase IQ, some clauses and keywords are ignored. For the full syntax of the Adaptive Server Enterprise CREATE INDEX statement, see the *Adaptive Server Enterprise Reference Manual, Volume 2: Commands*.

Adaptive Server Enterprise indexes can be either **clustered** or **nonclustered**. A clustered index almost always retrieves data faster than a nonclustered index. Only one clustered index is permitted per table.

Sybase IQ does not support clustered indexes. The CLUSTERED and NONCLUSTERED keywords are allowed by SQL Anywhere, but are ignored by Sybase IQ. If no *index-type* is specified, Sybase IQ creates an HG index on the specified column(s).

Sybase IQ does not permit the DESC keyword.

Sybase IQ also allows, by ignoring, the following keywords:

- FILLFACTOR
- IGNORE_DUP_KEY
- SORTED_DATA
- IGNORE_DUP_ROW
- ALLOW_DUP_ROW
- ON

Index names must be unique on a given table for both Sybase IQ and Adaptive Server Enterprise.

Permissions

Must be the owner of the table or have DBA authority.

See also

CREATE JOIN INDEX statement on page 481

DROP statement on page 533

INDEX_PREFERENCE option on page 88

Chapter 6, "Using Sybase IQ Indexes," in the *Sybase IQ System Administration Guide*.

CREATE JOIN INDEX statement

Description

Creates a join index, which defines a group of tables that are prejoined through specific columns, to improve performance of queries using tables in a join operation.

Syntax

CREATE JOIN INDEX *join-index-name* **FOR** *join-clause*

Parameters

join-clause:
[( ] *join-expression join-type join-expression*
[ON *search-condition*] [ )]

join-expression:
{ *table-name* | *join-clause* }

join-type:
[NATURAL] FULL [OUTER] JOIN

search-condition:
[( ] *search-expression* [AND *search-expression*] [ )]

search-expression:
[( ] [*table-name.*] *column-name* = [*table-name.*] *column-name* [ )]

Examples

This example creates a join index between the department and employee tables using the dept_id column, which is the primary key for department and foreign key for employee.

```
CREATE JOIN INDEX emp_dept_join
FOR department FULL OUTER JOIN employee
ON department.dept_id = employee.dept_id
```

Usage

CREATE JOIN INDEX creates a join index on the specified columns of the named tables. Once a join index is created, it is never referenced again except to delete it using DROP JOIN INDEX or to synchronize it using SYNCHRONIZE JOIN INDEX. This statement supports joins only of type FULL OUTER; the OUTER keyword is optional.

Note In a Sybase IQ multiplex, always perform CREATE JOIN INDEX in single-node mode on the write server, then synchronize query servers. CREATE JOIN INDEX returns an error instead of propagating from write server to query server.

ON References only columns from two tables. One set of columns must be from a single table in the left subtree and the other set of columns must be from a table in the right subtree. The only predicates supported are equijoin predicates. Sybase IQ does not allow single-variable predicates, intra-column comparisons, or nonequality joins.

Join index columns must have identical data type, precision, and scale.

To specify a multipart key, include more than one predicate linking the two tables connected by a logical AND. A disjunct ON clause is not supported; that is, Sybase IQ does not permit a logical OR of join predicates. Also, the ON clause does not accept a standard WHERE clause, so you cannot specify an alias.

You can use the NATURAL keyword instead of an ON clause. A NATURAL join is one that pairs columns up by name and implies an equijoin. If the NATURAL join generates predicates involving more than one pair of tables, CREATE JOIN INDEX returns an error. You can specify NATURAL or ON, but not both.

CREATE JOIN INDEX looks for a primary-key-to-foreign-key relationship in the tables to determine the direction of the one-to-many relationship. (The direction of a one-to-one relationship is not important.) The primary key is always the “one” and the foreign key is always the “many”. If such information is not defined, Sybase IQ assumes the subtree on the left is the “one” while the subtree on the right is the “many”. If the opposite is true, CREATE JOIN INDEX returns an error.

Note Query optimizations for all joins rely heavily on underlying primary keys. They do not require foreign keys. However, you can benefit from using foreign keys. Sybase IQ enforces foreign keys if you set up your loads to check for primary key-foreign key relationships.

Join index tables must be Sybase IQ base tables. They cannot be temporary tables, remote tables, or proxy tables.

Multicolumn indexes on base tables are *not* replicated in join indexes created using those base tables.

A star-join index is one in which a single table at the center of the star is joined to multiple tables in a one-to-many relationship. To define a star-join index, you must define single-column key and primary keys, and then use the key join syntax in the CREATE JOIN INDEX statement. Sybase IQ does not support star-join indexes that use multiple join key columns for any join.

The FLOAT_AS_DOUBLE option, which defaults to OFF, must be set ON for JDBC and client connections for CREATE JOIN INDEX statements to succeed.

If a join column is a REAL datatype, however, you must set FLOAT_AS_DOUBLE to OFF when creating join indexes, or an error occurs. Issues might also result from using inexact numerics for join columns.

Note You must explicitly grant permissions on the underlying "join virtual table" to other users in your group before they can manipulate tables in the join. For information on granting privileges on the join virtual table, see "Inserting or deleting from tables in a join index" in Chapter 6, "Using Sybase IQ Indexes" in the *Sybase IQ System Administration Guide*.

Side effects

Automatic commit.

Standards

- **SQL92** Intermediate-level feature.
- **Sybase** Not supported by Adaptive Server Enterprise.

Permissions

Must have DBA authority or have RESOURCE authority and be the owner of all tables involved in the join.

See also

CREATE INDEX statement on page 473

CREATE TABLE statement on page 499

Chapter 6, "Using Sybase IQ Indexes," in *Sybase IQ System Administration Guide*.

CREATE MESSAGE statement [T-SQL]

Description

Adds a user-defined message to the SYSUSERMESSAGES system table for use by PRINT and RAISERROR statements.

Syntax

CREATE MESSAGE *message-number*
... **AS** *'message-text'*

Usage

CREATE MESSAGE associates a message number with a message string. The message number can be used in PRINT and RAISERROR statements.

- **message_number** The message number of the message to add. The message number for a user-defined message must be 20000 or greater.
- **message_text** The text of the message to add. The maximum length is 255 bytes. PRINT and RAISERROR recognize placeholders in the message text to print out. A single message can contain up to 20 unique placeholders in any order. These placeholders are replaced with the formatted contents of any arguments that follow the message when the text of the message is sent to the client.

  Placeholders are numbered to allow reordering of the arguments when translating a message to a language with a different grammatical structure. A placeholder for an argument appears as "%nn!"—a percent sign (%), followed by an integer from 1 to 20, followed by an exclamation mark (!)—where the integer represents the position of the argument in the argument list, "%1!" is the first argument, "%2!" is the second argument, and so on.

There is no parameter corresponding to the *language* argument for sp_addmessage.

Side effects

Automatic commit.

Standards

- **SQL92** Vendor extension.
- **Sybase** The functionality of CREATE MESSAGE is provided by the sp_addmessage procedure in Adaptive Server Enterprise.

Permissions

Must have RESOURCE authority.

See also

PRINT statement [T-SQL] on page 613

RAISERROR statement [T-SQL] on page 616

CREATE PROCEDURE statement

Description

Creates a new procedure in the database.

Syntax

CREATE PROCEDURE [*owner*.]*procedure-name* ([*parameter* , ...]) {
[**RESULT** (*result-column*, ...) | **NO RESULT SET**] [**ON EXCEPTION RESUME**] *compound statement* | **AT** *location-string* | | [**DYNAMIC RESULT SETS** *integer-expression*]
[**EXTERNAL NAME** *java-call* **LANGUAGE JAVA**]
}

Parameters

parameter:
parameter_mode parameter-name data-type [DEFAULT *expression*]
| SQLCODE
| SQLSTATE

parameter_mode:
IN | OUT | INOUT

result-column:
column-name data-type

library-call:
'*function-name@library.dll*; ...'

java-call:
'[*package-name*.]*class-name*.*method-name method-signature*'

method-signature:
([*field-descriptor*,...]) *return-descriptor*

field-descriptor | *return-descriptor*:
Z | B | S | I | J | F | D | C | V | [*descriptor* | L*class-name*;

Examples

- This procedure uses a case statement to classify the results of a query.

```
CREATE PROCEDURE ProductType (IN product_id INT, OUT
type CHAR(10))
BEGIN
    DECLARE prod_name CHAR(20) ;
    SELECT name INTO prod_name FROM "DBA"."product"
    WHERE id = product_id;
    CASE prod_name
    WHEN 'Tee Shirt' THEN
       SET type = 'Shirt'
    WHEN 'Sweatshirt' THEN
       SET type = 'Shirt'
    WHEN 'Baseball Cap' THEN
       SET type = 'Hat'
    WHEN 'Visor' THEN
```

```
        SET type = 'Hat'
    WHEN 'Shorts' THEN
        SET type = 'Shorts'
    ELSE
        SET type = 'UNKNOWN'
    END CASE ;
END
```

- This procedure uses a cursor and loops over the rows of the cursor to return a single value.

```
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35),
OUT TopValue INT)
BEGIN
    DECLARE err_notfound EXCEPTION
    FOR SQLSTATE '02000' ;
    DECLARE curThisCust CURSOR FOR
    SELECT company_name, CAST(
sum(sales_order_items.quantity *
    product.unit_price) AS INTEGER) VALUE
    FROM customer
    LEFT OUTER JOIN sales_order
    LEFT OUTER JOIN sales_order_items
    LEFT OUTER JOIN product
    GROUP BY company_name ;


    DECLARE ThisValue INT ;
    DECLARE ThisCompany CHAR(35) ;
    SET TopValue = 0 ;
    OPEN curThisCust ;
    CustomerLoop:
    LOOP
        FETCH NEXT curThisCust
        INTO ThisCompany, ThisValue ;
        IF SQLSTATE = err_notfound THEN
            LEAVE CustomerLoop ;
        END IF ;
        IF ThisValue > TopValue THEN
            SET TopValue = ThisValue ;
            SET TopCompany = ThisCompany ;
            END IF ;
    END LOOP CustomerLoop ;
    CLOSE curThisCust ;
END
```

Usage

CREATE PROCEDURE creates a procedure in the database. Users with DBA authority can create procedures for other users by specifying an owner. A procedure is invoked with a CALL statement.

The body of a procedure consists of a compound statement. For information on compound statements, see BEGIN... END statement on page 422.

CREATE PROCEDURE Parameter names must conform to the rules for other database identifiers such as column names. They must be a valid SQL data type (see Chapter 4, "SQL Data Types"), and must be prefixed by IN, OUT or INOUT. The keywords have the following meanings:

- **IN** The parameter is an expression that provides a value to the procedure.
- **OUT** The parameter is a variable that could be given a value by the procedure.
- **INOUT** The parameter is a variable that provides a value to the procedure, and could be given a new value by the procedure.

When procedures are executed using CALL, not all parameters need to be specified. If a default value is provided in the CREATE PROCEDURE statement, missing parameters are assigned the default values. If an argument is not provided in the CALL statement, and no default is set, an error is given.

SQLSTATE and SQLCODE are special parameters that output the SQLSTATE or SQLCODE value when the procedure ends (they are OUT parameters). Whether or not a SQLSTATE and SQLCODE parameter is specified, the SQLSTATE and SQLCODE special values can always be checked immediately after a procedure call to test the return status of the procedure.

The SQLSTATE and SQLCODE special values are modified by the next SQL statement. Providing SQLSTATE or SQLCODE as procedure arguments allows the return code to be stored in a variable.

RESULT The RESULT clause declares the number and type of columns in the result set. The parenthesized list following the RESULT keyword defines the result column names and types. This information is returned by the Embedded SQL DESCRIBE or by ODBC SQLDescribeCol when a CALL statement is being described. Allowed data types are listed in Chapter 4, "SQL Data Types."

For more information on returning result sets from procedures, see Chapter 8, "Using Procedures and Batches" in the *Sybase IQ System Administration Guide*.

Some procedures can return more than one result set, with different numbers of columns, depending on how they are executed. For example, the following procedure returns two columns under some circumstances, and one in others.

```
CREATE PROCEDURE names( IN formal char(1))
BEGIN
    IF formal = 'n' THEN
       SELECT emp_fname
       FROM employee
    ELSE
       SELECT emp_lname,emp_fname
       FROM employee
    END IF
END
```

Procedures with variable result sets must be written without a RESULT clause, or in Transact-SQL. Their use is subject to the following limitations:

- **Embedded SQL** You must DESCRIBE the procedure call after the cursor for the result set is opened, but before any rows are returned, in order to get the proper shape of result set. The CURSOR *cursor-name* clause on the DESCRIBE statement is required.
- **ODBC** Variable result-set procedures can be used by ODBC applications. The proper description of the result sets is carried out by the ODBC driver.
- **Open Client applications** Variable result-set procedures can be used by Open Client applications.

If your procedure returns only one result set, use a RESULT clause. The presence of this clause prevents ODBC and Open Client applications from describing the result set again after a cursor is open.

To handle multiple result sets, ODBC must describe the currently executing cursor, not the procedure's defined result set. Therefore, ODBC does not always describe column names as defined in the RESULT clause of the procedure definition. To avoid this problem, use column aliases in the SELECT statement that generates the result set.

NO RESULT SET This clause declares that this procedure returns no result set. This is useful when an external environment needs to know that a procedure does not return a result set.

ON EXCEPTION RESUME This clause enables Transact-SQL -like error handling to be used within a Watcom-SQL syntax procedure.

If you use ON EXCEPTION RESUME, the procedure takes an action that depends on the setting of the ON_TSQL_ERROR option. If ON_TSQL_ERROR is set to CONDITIONAL (which is the default) the execution continues if the next statement handles the error; otherwise, it exits.

Error-handling statements include the following:

- IF
- SELECT @variable =
- CASE
- LOOP
- LEAVE
- CONTINUE
- CALL
- EXECUTE
- SIGNAL
- RESIGNAL
- DECLARE
- SET VARIABLE

Do not use explicit error-handling code with an ON EXCEPTION RESUME clause.

For more information, see "ON_TSQL_ERROR option [TSQL]" on page 128.

AT location-string Create a **proxy stored procedure** on the current database for a remote procedure specified by *location-string*. The AT clause supports the semicolon (;) as a field delimiter in *location-string*. If no semicolon is present, a period is the field delimiter. This allows file names and extensions to be used in the database and owner fields.

For example, the following statement creates the proxy procedure remotewho that calls the dbo.sp_who procedure on the master database of the bostonase server:

```
CREATE PROCEDURE remotewho ()
AT 'bostonase.master.dbo.sp_who'
```

Remote procedures can return only up to 254 characters in output variables.

For information on remote servers, see CREATE SERVER statement on page 494. For information on using remote procedures, see the section "Using remote procedure calls (RPCs)" in Chapter 16, "Accessing Remote Data" in the *Sybase IQ System Administration Guide*.

DYNAMIC RESULT SETS This clause is for use with procedures that are wrappers around Java methods. If the DYNAMIC RESULT SETS clause is not provided, it is assumed that the method returns no result set.

EXTERNAL NAME LANGUAGE JAVA A procedure that uses EXTERNAL NAME with a LANGUAGE JAVA clause is a wrapper around a Java method.

If the number of parameters is less than the number indicated in the method-signature, the difference must equal the number specified in DYNAMIC RESULT SETS, and each parameter in the method signature in excess of those in the procedure parameter list must have a method signature of [Ljava/sql/ResultSet;.

Java method signatures A Java method signature is a compact character representation of the types of the parameters and the type of the return value.

The meanings of *field-descriptor* and *return-descriptor* are listed in Table 6-7.

Table 6-7: Java method signatures

Field type	Java data type
B	byte
C	char
D	double
F	float
I	int
J	long
L*class-name*;	an instance of the class *class-name*. The class name must be fully qualified, and any dot in the name must be replaced by a /. For example, java/lang/String
S	short
V	void
Z	boolean
[	use one for each dimension of an array

For example:

```
double some_method(
  boolean a,
  int b,
  java.math.BigDecimal c,
  byte [][] d,
  java.sql.ResultSet[] d ) {
}
```

would have the following signature:

```
'(ZILjava/math/BigDecimal;[[B[Ljava/sql/ResultSet;)D'
```

Note As procedures are dropped and created, databases created prior to Sybase IQ 12.6 may eventually reach the maximum proc_id limit of 32767, causing CREATE PROCEDURE to return an "Item already exists" error in Sybase IQ 12.6. For workaround, see "Insufficient procedure identifiers," *Sybase IQ Troubleshooting and Recovery Guide*.

Side effects

Automatic commit.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** The Transact-SQL CREATE PROCEDURE statement is different.
- **SQLJ** The syntax extensions for Java result sets are as specified in the proposed SQLJ1 standard.

Permissions

Must have RESOURCE authority. For external procedures, must have DBA authority.

See also

BEGIN... END statement on page 422

CALL statement on page 429

DROP statement on page 533

EXECUTE IMMEDIATE statement [ESQL] [SP] on page 544

GRANT statement on page 559

"Copy Definition utility (defncopy)" in Chapter 3, "Database Administration Utilities" of the *Sybase IQ Utility Guide*

CREATE PROCEDURE statement [T-SQL]

Description

Creates a new procedure in the database in a manner compatible with Adaptive Server Enterprise.

Syntax

The following subset of the Transact-SQL CREATE PROCEDURE statement is supported in Sybase IQ.

CREATE PROCEDURE [*owner*.]*procedure_name*
... [[( ] *@parameter_name data-type* [= *default*] [**OUTPUT**] [, ..] [ )]]
...[**WITH RECOMPILE**]

...**AS**
...statement-list

Usage

The following differences between Transact-SQL and Sybase IQ statements are listed to help those writing in both dialects.

- **Variable names prefixed by @** The "@" sign denotes a Transact-SQL variable name, while Sybase IQ variables can be any valid identifier, and the @ prefix is optional.
- **Input and output parameters** Sybase IQ procedure parameters are specified as IN, OUT, or INOUT, while Transact-SQL procedure parameters are INPUT parameters by default or can be specified as OUTPUT. Those parameters that would be declared as INOUT or as OUT in Sybase IQ should be declared with OUTPUT in Transact-SQL.
- **Parameter default values** Sybase IQ procedure parameters are given a default value using the keyword DEFAULT, while Transact-SQL uses an equality sign (=) to provide the default value.
- **Returning result sets** Sybase IQ uses a RESULT clause to specify returned result sets. In Transact-SQL procedures, the column names or alias names of the first query are returned to the calling environment.

```
CREATE PROCEDURE showdept @deptname varchar(30)
AS
    SELECT employee.emp_lname, employee.emp_fname
    FROM department, employee
    WHERE department.dept_name = @deptname
    AND department.dept_id = employee.dept_id
```

The following is the corresponding Sybase IQ procedure:

```
CREATE PROCEDURE showdept(in deptname
          varchar(30) )
RESULT ( lastname char(20), firstname char(20))
ON EXCEPTION RESUME
BEGIN
    SELECT employee.emp_lname, employee.emp_fname
    FROM department, employee
    WHERE department.dept_name = deptname
    AND department.dept_id = employee.dept_id
END
```

- **Procedure body** The body of a Transact-SQL procedure is a list of Transact-SQL statements prefixed by the AS keyword. The body of a Sybase IQ procedure is a compound statement, bracketed by BEGIN and END keywords.

Side effects

Automatic commit.

Standards

- **SQL92** Transact-SQL extension.
- **Sybase** Sybase IQ supports a subset of the Adaptive Server Enterprise CREATE PROCEDURE statement syntax.

  If the Transact-SQL WITH RECOMPILE optional clause is supplied, it is ignored. Adaptive Server Anywhere always recompiles procedures the first time they are executed after a database is started, and stores the compiled procedure until the database is stopped.

  Groups of procedures are not supported.

Permissions

Must have RESOURCE authority.

See also

CREATE PROCEDURE statement on page 485

"Copy Definition utility (defncopy)" in Chapter 3, "Database Administration Utilities" of the *Sybase IQ Utility Guide*

CREATE SCHEMA statement

Description

Creates a schema, which is a collection of tables, views, and permissions and their associated permissions, for a database user.

Syntax

CREATE SCHEMA AUTHORIZATION *userid*
... [{ *create-table-statement*
| *create-view-statement*
| *grant-statement* }]...

Usage

The *userid* must be the user ID of the current connection. You cannot create a schema for another user. The user ID is not case sensitive.

If any of the statements in the CREATE SCHEMA statement fail, the entire CREATE SCHEMA statement is rolled back.

CREATE SCHEMA statement is simply a way to collect individual CREATE and GRANT statements into one operation. There is no SCHEMA database object created in the database, and to drop the objects you must use individual DROP TABLE or DROP VIEW statements. To revoke permissions, use a REVOKE statement for each permission granted.

Note The CREATE SCHEMA statement is invalid on an active multiplex.

Individual CREATE or GRANT statements are not separated by statement delimiters. The statement delimiter marks the end of the CREATE SCHEMA statement itself.

The individual CREATE or GRANT statements must be ordered such that the objects are created before permissions are granted on them.

Although you can currently create more than one schema for a user, this is not recommended, and might not be supported in future releases.

Side effects

Automatic commit.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Sybase IQ does not support the use of REVOKE statements within the CREATE SCHEMA statement, and does not allow its use within Transact-SQL batches or procedures.

Permissions

Must have RESOURCE authority.

See also

CREATE TABLE statement on page 499

CREATE VIEW statement on page 512

GRANT statement on page 559

CREATE SERVER statement

Description

Adds a server to the sysservers table.

Syntax

CREATE SERVER *server-name*
CLASS '*server-class*'
USING '*connection-info*'
[**READ ONLY**]

Parameters

server-class:
{ ASAJDBC| ASEJDBC
| ASAODBC | ASEODBC
| DB2ODBC | MSSODBC
| ORAODBC | ODBC}

connection-info:
{ *machine-name:port-number* [*/dbname*] | *data-source-name* }

Examples

- Creates a remote server for the JDBC-based Adaptive Server named ase_prod. Its machine name is “banana” and port number is 3025.

```
CREATE SERVER ase_prod
CLASS 'asejdbc'
USING 'banana:3025'
```

- Creates an Adaptive Server Anywhere remote server named testasa, located on the machine "apple," and listening on port number 2638. Use:

```
CREATE SERVER testasa
CLASS 'asajdbc'
USING 'apple:2638'
```

- Create a remote server for the Oracle server named oracle723. Its ODBC Data Source Name is "oracle723."

```
CREATE SERVER oracle723
CLASS 'oraodbc'
USING 'oracle723'
```

Usage

CREATE SERVER defines a remote server from the Sybase IQ catalogs.

For more information on server classes and how to configure a server, see Chapter 17, "Server Classes for Remote Data Access" in the *Sybase IQ System Administration Guide*.

USING clause If a JDBC-based server class is used, the USING clause is hostname:port-number [/dbname] where:

- **hostname** Is the machine on which the remote server runs.
- **portnumber** Is the TCP/IP port number on which the remote server listens. The default port number for Sybase IQ and Adaptive Server Anywhere is 2638.
- **dbname** For Adaptive Server Anywhere remote servers, if you do not specify a *dbname*, the default database is used. For Adaptive Server Enterprise, the default is the master database, and an alternative to using *dbname* is to another database by some other means (for example, in the FORWARD TO statement).

For more information, see "JDBC-based server classes" in the *Sybase IQ System Administration Guide*.

If an ODBC-based server class is used, the USING clause is the *data-source-name*. The data-source-name is the ODBC Data Source Name.

READ ONLY The READ ONLY clause specifies that the remote server is a read-only data source. Any update request is rejected by Sybase IQ.

Side effects

Automatic commit.

Standards
- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions Must have RESOURCE authority.

See also "ALTER SERVER statement" on page 405

"DROP SERVER statement" on page 538

CREATE SERVICE statement

Description Permits a database server to act as a Web server.

Syntax
CREATE SERVICE *service-name*
TYPE *service-type-string*
[*attributes*] [
AS *statement*]

Parameters
attributes:
[AUTHORIZATION { ON
| OFF }] [SECURE
{ ON | OFF }] [USER { *user-name* | NULL } [ ] URL [PATH] { ON | OFF |
ELEMENTS }] [USING { *SOAP-prefix* | NULL }]

service-type-string:
{ 'RAW ' | 'HTML ' |
'XML ' |
'SOAP ' |
' DISH ' }

service-name Web service names may be any sequence of alphanumeric characters or "/", "-", "_", ".", "!", "~", "*", "'", "(", or ")", except that the first character cannot begin with a slash (/) and the name cannot contain two or more consecutive slash characters.

service-type-string Identifies the type of the service. The type must be one of the listed service types. There is no default value.

AUTHORIZATION clause Determines whether users must specify a user name and password when connecting to the service. If authorization is OFF, the AS clause is required and a single user must be identified by the USER clause. All requests are run using that user's account and permissions.

If authorization is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name using the USER clause. If the user name is NULL, all known users can access the service.

The default value is ON. Sybase recommends that production systems be run with authorization turned on and that you grant permission to use the service by adding users to a group.

SECURE clause Indicates whether unsecure connections are accepted. ON indicates that only HTTPS connections are to be accepted. Service requests received on the HTTP port are automatically redirected to the HTTPS port. If set to OFF, both HTTP and HTTPS connections are accepted. The default value is OFF.

USER clause If authorization is disabled, this parameter becomes mandatory and specifies the user id used to execute all service requests. If authorization is enabled (the default), this optional clause identifies the user or group permitted access to the service. The default value is NULL, which grants access to all users.

URL clause Determines whether URI paths are accepted and, if so, how they are processed. OFF indicates that nothing must follow the service name in a URI request. ON indicates that the remainder of the URI is interpreted as the value of a variable named url. ELEMENTS indicates that the remainder of the URI path is to be split at the slash characters into a list of up to 10 elements. The values are assigned to variables named url plus a numeric suffix of between 1 and 10; for example, the first three variable names are url1, url2, and url3. If fewer than 10 values are supplied, the remaining variables are set to NULL. If the service name ends with the character /, then URL must be set to OFF. The default value is OFF.

USING clause This clause applies only to DISH services. The parameter specifies a name prefix. Only SOAP services whose names begin with this prefix are handled.

statement If the statement is NULL, the URI must specify the statement to be executed. Otherwise, the specified SQL statement is the only one that can be executed through the service. The statement is mandatory for SOAP services, and ignored for DISH services. The default value is NULL.

Sybase strongly recommends that all services run in production systems define a statement. The statement can be NULL only if authorization is enabled.

RAW The result set is sent to the client without any further formatting. You can produce formatted documents by generating the required tags explicitly within your procedure, as demonstrated in an example, below.

HTML The result set of a statement or procedure is automatically formatted into an HTML document that contains a table.

XML The result set is assumed to be in XML format. If it is not already so, it is automatically converted to XML RAW format.

SOAP The request must be a valid Simple Object Access Protocol, or SOAP, request. The result set is automatically formatted as a SOAP response. For more information about the SOAP standards, see www.w3.org/TR/SOAP at http://www.w3.org/TR/SOAP.

DISH A Determine SOAP Handler, or DISH, service acts as a proxy for one or more SOAP services. In use, it acts as a container that holds and provides access to a number of SOAP services. A Web Services Description Language (WSDL) file is automatically generated for each of the included SOAP services. The included SOAP services are identified by a common prefix, which must be specified in the USING clause.

The create service statement causes the database server to act as a web server. A new entry is created in the SYSWEBSERVICE system table.

Examples

Example 1 To set up a Web server quickly, start a database server with the `-xs` switch, then execute the following statement:

```
CREATE SERVICE tables TYPE 'HTML'
AUTHORIZATION OFF    USER DBA
AS SELECT * FROM SYS.SYSTABLE
```

After executing this statement, use any Web browser to open the URL *http://localhost/tables.*

Example 2 The following example demonstrates how to write a Hello World program.

```
CREATE PROCEDURE hello_world_proc RESULT (html_doc long
varchar) BEGIN   CALL dbo.sa_set_http_header( 'Content-
```

```
Type', 'text/html' );    SELECT '<html>\n'        ||
'<head><title>Hello World</title></head>\n'        ||
'<body>\n'       || '<h1>Hello World!</h1>\n'       ||
'</body>\n'        || '</html>\n'; END;

CREATE SERVICE hello_world TYPE 'RAW' AUTHORIZATION OFF
USER DBA AS CALL hello_world_proc;
```

Usage

The create service statement causes the database server to act as a web server. A new entry is created in the SYSWEBSERVICE system table.

Standards

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

Permissions

Must have DBA authority.

See also

"ALTER SERVICE statement" on page 407

"DROP SERVICE statement" on page 539

"Using the Built-in Web Server" in *Adaptive Server Anywhere Database Administration Guide*.

CREATE TABLE statement

Description

Creates a new table in the database or on a remote server.

Syntax

CREATE [**GLOBAL TEMPORARY**] **TABLE** [*owner*.]*table-name*
... (*column-definition* [*column-constraint*]...
[, *column-definition* [*column-constraint*]...]
[, *table-constraint*]...)
... [{ **IN** | **ON** } *dbspace-name*]
... [**ON COMMIT** { **DELETE** | **PRESERVE** } **ROWS**
| **NOT TRANSACTIONAL**]
[**AT** *location-string*]

Parameters

column-definition:
column-name data-type [[NOT] NULL] [DEFAULT *default-value* | IDENTITY]

default-value:
special-value
| *string*
| *global variable*
| [-] *number*
| (*constant-expression*)
| *built-in-function*(*constant-expression*)
| AUTOINCREMENT
| CURRENT DATABASE
| CURRENT REMOTE USER
| NULL
| TIMESTAMP
| LAST USER

special-value:
CURRENT { DATE | TIME | TIMESTAMP | USER | PUBLISHER }
| USER

column-constraint:
{ UNIQUE
| PRIMARY KEY
| REFERENCES *table-name* [(*column-name*)] [*actions*]
| CHECK (*condition*)
| IQ UNIQUE (*integer*) }

table-constraint:
{ UNIQUE (*column-name* [, *column-name*]...)
| PRIMARY KEY (*column-name* [, *column-name*]...)
| CHECK (*condition*)
| *foreign-key-constraint* }

foreign-key-constraint:
FOREIGN KEY [*role-name*]
[(*column-name* [, *column-name*]...)]
... REFERENCES *table-name* [(*column-name* [, *column-name*]...)]
... [*action*] [

action:
ON { UPDATE | DELETE { RESTRICT }

location-string:
{ *remote-server-name*.[*db-name*].[*owner*].*object-name*
| *remote-server-name*;[*db-name*];[*owner*];*object-name* }

Examples

- Creates a table for a library database to hold book information:

```
CREATE TABLE library_books (
isbn CHAR(20)       PRIMARY KEY IQ UNIQUE (150000),
copyright_date      DATE,
title               CHAR(100),
author              CHAR(50)
)
```

- Creates a table for a library database to hold information on borrowed books:

```
CREATE TABLE borrowed_book (
date_borrowed DATE NOT NULL,
date_returned DATE,
book        CHAR(20)
            REFERENCES library_books (isbn),
CHECK( date_returned >= date_borrowed )
)
```

- Creates a table named t1at the remote server SERVER_A and create a proxy table named t1 that is mapped to the remote table:

```
CREATE TABLE t1
( a  INT,
  b  CHAR(10))
AT 'SERVER_A.db1.joe.t1'
```

- Creates a table named tab1 that contains a column c1 with a default value of the special constant LAST USER:

```
CREATE TABLE tab1(c1 CHAR(20) LAST USER)
```

Usage

You can create a table for another user by specifying an owner name. If GLOBAL TEMPORARY is not specified, the table is referred to as a base table. Otherwise, the table is a temporary table.

A created global temporary table exists in the database like a base table and remains in the database until it is explicitly removed by a DROP TABLE statement. The rows in a temporary table are visible only to the connection that inserted the rows. Multiple connections from the same or different applications can use the same temporary table at the same time and each connection sees only its own rows. A given connection inherits the schema of a global temporary table as it exists when the connection first refers to the table. The rows of a temporary table are deleted when the connection ends.

When you create a local temporary table, omit the owner specification. If you specify an owner when creating a temporary table, as, for example, with `CREATE TABLE dbo.#temp(col1 int)`, a base table is incorrectly created.

You cannot use a temporary table to create a join index.

Also, you cannot update a base table that is part of any join index. If you do, you see the following error message:

```
-1000102 Cannot update table %2 because it is defined
in one or more join indexes
```

IN Specifies in which database file (dbspace) the table is to be created. You can specify SYSTEM with this clause to put either a permanent or temporary table in the Catalog Store. All other use of the IN clause is ignored. You *cannot* use this clause to place an IQ table in a particular dbspace. By default, all permanent tables are placed in the main IQ Store, and all temporary tables are placed in the Temporary IQ Store. Global temporary tables can never be in the IQ Store.

Note While executing CREATE TABLE statements propagated from a multiplex write server, Sybase IQ resolves conflicts by renaming any existing query server persistent objects that have the same names as proposed objects. See “Resolving static collisions” in Chapter 5, “Working with Database Objects” in the *Sybase IQ System Administration Guide*.

For more information about dbspaces, see CREATE DBSPACE statement on page 453.

ON COMMIT Allowed for temporary tables only. By default, the rows of a temporary table are deleted on COMMIT.

NOT TRANSACTIONAL Allowed only for temporary tables. A table created using NOT TRANSACTIONAL is not affected by either COMMIT or ROLLBACK.

The NOT TRANSACTIONAL clause provides performance improvements in some circumstances because operations on nontransactional temporary tables do not cause entries to be made in the rollback log. For example, NOT TRANSACTIONAL might be useful if procedures that use the temporary table are called repeatedly with no intervening COMMITs or ROLLBACKs.

The parenthesized list following the CREATE TABLE statement can contain the following clauses in any order:

AT Used to create a table at the remote location specified by *location-string*. The local table that is created is a proxy table that maps to the remote location. Tables used as proxy tables must have names of 30 characters or less. The AT clause supports the semicolon (;) as a delimiter. If a semicolon is present anywhere in the *location-string*, the semicolon is the field delimiter. If no semicolon is present, a period is the field delimiter. This allows file names and extensions to be used in the database and owner fields.

Semicolon field delimiters are used primarily with server classes not currently supported; however, you can also use them in situations where a period would also work as a field delimiter. For example, the following statement maps the table proxy_a to the Adaptive Server Anywhere database mydb on the remote server myasa:

```
CREATE TABLE proxy_a1
AT 'myasa;mydb;;a1'
```

Foreign-key definitions are ignored on remote tables. Foreign-key definitions on local tables that refer to remote tables are also ignored. Primary key definitions are sent to the remote server if the server supports primary keys.

column-definition Defines a column in the table. Allowable data types are described in Chapter 4, “SQL Data Types.” Two columns in the same table cannot have the same name. If NOT NULL is specified, or if the column is in a UNIQUE or PRIMARY KEY constraint, the column cannot contain any NULL values. You can create up to 45,000 columns; however, there might be performance penalties with more than 10,000 columns in a table.

- **DEFAULT default-value** When defining a column for a table, you can specify a default value for the column using the DEFAULT keyword in the CREATE TABLE (and ALTER TABLE) statement. If a DEFAULT value is specified for a column, this DEFAULT value is used as the value of the column in any INSERT (or LOAD) statement that does not specify a value for the column.

  For detailed information on the use of column DEFAULT values, see “Using column defaults” in Chapter 9, “Ensuring Data Integrity” in the *Sybase IQ System Administration Guide*.

- **DEFAULT AUTOINCREMENT** The value of the DEFAULT AUTOINCREMENT column uniquely identifies every row in a table. Columns of this type are also known as IDENTITY columns, for compatibility with Adaptive Server Enterprise. The IDENTITY/DEFAULT AUTOINCREMENT column stores sequential numbers that are automatically generated during inserts and updates. When using IDENTITY or DEFAULT AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type, with scale 0. The column value might also be NULL. You must qualify the specified tablename with the owner name.

  ON inserts into the table. If a value is not specified for the IDENTITY/DEFAULT AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column, it is used; if the specified value is not larger than the current maximum value for the column, that value is used as a starting point for subsequent inserts.

  Deleting rows does not decrement the IDENTITY/AUTOINCREMENT counter. Gaps created by deleting rows can only be filled by explicit assignment when using an insert. The database option IDENTITY_INSERT must be set to the table name to perform an insert into an IDENTITY/AUTOINCREMENT column.

  For example, the following creates a table with an IDENTITY column and explicitly adds some data to it:

  ```
  CREATE TABLE mytable(c1 INT IDENTITY);
  SET TEMPORARY OPTION IDENTITY_INSERT =
  "DBA".mytable;
  INSERT INTO mytable VALUES(5);
  ```

  After an explicit insert of a row number less then the maximum, subsequent rows without explicit assignment are still automatically incremented with a value of one greater than the previous maximum.

  You can find the most recently inserted value of the column by inspecting the @@identity global variable.

- **IDENTITY** A Transact-SQL-compatible alternative to using the AUTOINCREMENT default. In Sybase IQ, the identity column may be created using either the IDENTITY or the DEFAULT AUTOINCREMENT clause.

table-constraint Helps ensure the integrity of data in the database. There are four types of integrity constraints:

- **UNIQUE constraint** Identifies one or more columns that uniquely identify each row in the table. No two rows in the table can have the same values in all the named columns. A table may have more than one unique constraint.
- **PRIMARY KEY constraint** Is the same as a UNIQUE constraint except that a table can have only one primary-key constraint. *You cannot specify the PRIMARY KEY and UNIQUE constraints for the same column*. The primary key usually identifies the best identifier for a row. For example, the customer number might be the primary key for the customer table.
- **FOREIGN KEY constraint** Restricts the values for a set of columns to match the values in a primary key or uniqueness constraint of another table. For example, a foreign-key constraint could be used to ensure that a customer number in an invoice table corresponds to a customer number in the customer table.

Note You cannot create foreign-key constraints on local temporary tables. Global temporary tables must be created with ON COMMIT PRESERVE ROWS.

- **CHECK constraint** Allows arbitrary conditions to be verified. For example, a check constraint could be used to ensure that a column called Gender contains only the values male or female. No row in a table is allowed to violate a constraint. If an INSERT or UPDATE statement would cause a row to violate a constraint, the operation is not permitted and the effects of the statement are undone.

  Column identifiers in column check constraints that start with the symbol '@' are placeholders for the actual column name. Thus a statement of the form:

  ```
  CREATE TABLE t1(c1 INTEGER CHECK (@foo < 5))
  ```

  is exactly the same as the following statement:

  ```
  CREATE TABLE t1(c1 INTEGER CHECK (c1 < 5))
  ```

  Column identifiers appearing in table check constraints that start with the symbol '@'are *not* placeholders.

If a statement would cause changes to the database that would violate an integrity constraint, the statement is effectively not executed and an error is reported. (*Effectively* means that any changes made by the statement before the error was detected are undone.)

Sybase IQ enforces single-column UNIQUE constraints by creating an HG index for that column.

Note You cannot define a column with a BIT data type as a UNIQUE or PRIMARY KEY constraint. Also, the default for columns of BIT data type is to not allow NULL values; you can change this by explicitly defining the column as allowing NULL values.

---

column-constraint Restricts the values the column can hold. Column and table constraints help ensure the integrity of data in the database. If a statement would cause a violation of a constraint, execution of the statement does not complete, any changes made by the statement before error detection are undone, and an error is reported. Column constraints are abbreviations for the corresponding table constraints. For example, the following are equivalent:

```
CREATE TABLE Product (
    product_num integer UNIQUE
)
CREATE TABLE Product (
    product_num integer,
    UNIQUE ( product_num )
)
```

Column constraints are normally used unless the constraint references more than one column in the table. In these cases, a table constraint must be used.

IQ UNIQUE constraint This constraint can be specified for columns only. IQ UNIQUE defines the cardinality of the column, and it is used to optimize the indexes internally. The default value is 0, which gives IQ no information for optimizing the default index. The IQ UNIQUE constraint should be applied if the expected distinct count (the number of unique values) for the column is less than or equal to 65536. This allows Sybase IQ to optimize storage of this column's data.

When the MINIMIZE_STORAGE option is ON (the default for new databases is OFF), it is equivalent to specifying IQ UNIQUE 255 for every newly created column, and there is no need to specify IQ UNIQUE except for columns with more than 65536 unique values. For related information, see “Optimizing storage and query performance,” *Sybase IQ System Administration Guide*.

Integrity constraints

UNIQUE or UNIQUE (column-name, ...) No two rows in the table can have the same values in all the named columns. A table may have more than one unique constraint.

There is a difference between a **unique constraint** and a **unique index**. Columns of a unique index are allowed to be NULL, while columns in a unique constraint are not. A foreign key can reference either a primary key or a column with a unique constraint, but not a unique index, because it can include multiple instances of NULL.

PRIMARY KEY or PRIMARY KEY (column-name, ...) The primary key for the table consists of the listed columns, and none of the named columns can contain any NULL values. Sybase IQ ensures that each row in the table has a unique primary key value. A table can have only one PRIMARY KEY.

When the second form is used (PRIMARY KEY followed by a list of columns), the primary key is created including the columns in the order in which they are defined, not the order in which they are listed.

When a column is designated as PRIMARY KEY, FOREIGN KEY, or UNIQUE, Sybase IQ creates a High_Group index for it automatically. For multicolumn primary keys, this index is on the primary key, not the individual columns. For best performance, you should also index each column with a HG or LF index separately.

REFERENCES primary-table-name [(primary-column-name)] This clause defines the column as a foreign key for a primary key or a unique constraint of a primary table. Normally, a foreign key would be for a primary key rather than an unique constraint. If a primary column name is specified, it must match a column in the primary table which is subject to a unique constraint or primary key constraint, and that constraint must consist of only that one column. Otherwise the foreign key references the primary key of the second table. Primary key and foreign key must have the same data type and the same precision, scale, and sign. Only a nonunique single-column HG index is created for a single-column foreign key. For a multicolumn foreign key, Sybase IQ creates a nonunique composite HG index. The maximum width of a multicolumn composite key for a unique or nonunique HG index is 1KB.

A temporary table cannot have a foreign key that references a base table and a base table cannot have a foreign key that references a temporary table. Local temporary tables cannot have or be referenced by a foreign key.

FOREIGN KEY [role-name] [(...)] REFERENCES primary-table-name [(...)] This clause defines foreign-key references to a primary key or a unique constraint in another table. Normally, a foreign key would be for a primary key rather than an unique constraint. (In this description, this other table is called the primary table.)

If the primary table column names are not specified, the primary table columns are the columns in the table's primary key. If foreign key column names are not specified, the foreign-key columns have the same names as the columns in the primary table. If foreign-key column names are specified, then the primary key column names must be specified, and the column names are paired according to position in the lists.

If the primary table is not the same as the foreign-key table, either the unique or primary key constraint must have been defined on the referenced key. Both referenced key and foreign key must have the same number of columns, of identical data type with the same sign, precision, and scale.

The value of the row's foreign key must appear as a candidate key value in one of the primary table's rows unless one or more of the columns in the foreign key contains nulls in a null allows foreign key column.

Any foreign-key column not explicitly defined is automatically created with the same data type as the corresponding column in the primary table. These automatically created columns cannot be part of the primary key of the foreign table. Thus, a column used in both a primary key and foreign key must be explicitly created.

role-name is the name of the foreign key. The main function of *role-name* is to distinguish two foreign keys to the same table. If no *role-name* is specified, the role name is assigned as follows:

1 If there is no foreign key with a *role-name* the same as the table name, the table name is assigned as the *role-name*.

2 If the table name is already taken, the *role-name* is the table name concatenated with a zero-padded 3-digit number unique to the table.

The referential integrity action defines the action to be taken to maintain foreign-key relationships in the database. Whenever a primary key value is changed or deleted from a database table, there may be corresponding foreign key values in other tables that should be modified in some way. You can specify an ON DELETE clause, followed by the RESTRICT clause:

RESTRICT Generates an error if you try to update or delete a primary key value while there are corresponding foreign keys elsewhere in the database. Generates an error if you try to update a foreign key so that you create new values unmatched by a candidate key. This is the default action, unless you specify that LOAD optionally reject rows that violate referential integrity.This enforces referential integrity at the statement level.

If you use CHECK ON COMMIT without specifying any actions, then RESTRICT is implied as an action for DELETE. Sybase IQ does not support CHECK ON COMMIT.

A global temporary table cannot have a foreign key that references a base table and a base table cannot have a foreign key that references a global temporary table. Local temporary tables cannot have or be referenced by a foreign key.

CHECK (condition) No row is allowed to fail the condition. If an INSERT statement would cause a row to fail the condition, the operation is not permitted and the effects of the statement are undone.

The change is rejected only if the condition is FALSE; in particular, the change is allowed if the condition is UNKNOWN. (See "NULL value" on page 218 and "Search conditions" on page 189 in Chapter 3, "SQL Language Elements" for more information about TRUE, FALSE, and UNKNOWN conditions.) CHECK condition is *not* enforced by Sybase IQ.

Note Sybase recommends that you not define referential integrity foreign key-primary key relationships in Sybase IQ unless you are certain there are no orphan foreign keys.

Remote tables

Foreign-key definitions are ignored on remote tables. Foreign-key definitions on local tables that refer to remote tables are also ignored. Primary-key definitions are sent to the remote server if the server supports it.

Side effects

Automatic commit.

Standards

- **SQL92** Entry-level feature.

  The following are vendor extensions:

  - The { IN | ON } *dbspace-name* clause
  - The ON COMMIT clause
  - Some of the default values

- **Sybase** Supported by Adaptive Server Enterprise, with some differences.

- **Temporary tables** You can create a temporary table by preceding the table name in a CREATE TABLE statement with a pound sign (#). These temporary tables are Sybase IQ declared temporary tables, which are available only in the current connection. For information about declared temporary tables, see DECLARE LOCAL TEMPORARY TABLE statement on page 523.
- **Physical placement** Physical placement of a table is carried out differently in Sybase IQ and in Adaptive Server Enterprise. The ON *segment-name* clause supported by Adaptive Server Enterprise is supported in Sybase IQ, but *segment-name* refers to an IQ dbspace.
- **Constraints** Sybase IQ does not support named constraints or named defaults, but does support user-defined data types that allow constraint and default definitions to be encapsulated in the data type definition. It also supports explicit defaults and CHECK conditions in the CREATE TABLE statement.
- **NULL default** By default, columns in Adaptive Server Enterprise default to NOT NULL, whereas in Sybase IQ the default setting is NULL, to allow NULL values. This setting can be controlled using the ALLOW_NULLS_BY_DEFAULT option. For information on this option, see ALLOW_NULLS_BY_DEFAULT option [TSQL] on page 40. You should explicitly specify NULL or NOT NULL to make your data definition statements transferable.

Permissions

Must have RESOURCE authority. To create a table for another user, you must have DBA authority.

See also

ALTER TABLE statement on page 409

CREATE DBSPACE statement on page 453

CREATE INDEX statement on page 473

DECLARE LOCAL TEMPORARY TABLE statement on page 523

DROP statement on page 533

MINIMIZE_STORAGE option on page 117

Chapter 5, “Working with Database Objects” in *Sybase IQ System Administration Guide*

CREATE VARIABLE statement

Description

Creates a SQL variable.

Syntax

CREATE VARIABLE *identifier data-type*

Examples

- The following code fragment inserts a large text value into the database:

```
EXEC SQL BEGIN DECLARE SECTION;
char buffer[5000];
EXEC SQL END DECLARE SECTION;
EXEC SQL CREATE VARIABLE hold_blob VARCHAR;
EXEC SQL SET hold_blob = '';
for(;;) {
    /* read some data into buffer ... */
    size = fread( buffer, 1, 5000, fp );
    if( size <= 0 ) break;
    /* add data to blob using concatenation
    Note that concatenation works for binary
    data too! */
    EXEC SQL SET hold_blob = hold_blob || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES ( 1, hold_blob );
EXEC SQL DROP VARIABLE hold_blob;
```

Usage

The CREATE VARIABLE statement creates a new variable of the specified data type. The variable contains the NULL value until it is assigned a different value by the SET VARIABLE statement.

A variable can be used in a SQL expression anywhere a column name is allowed. If a column name exists with the same name as the variable, the variable value is used.

Variables belong to the current connection, and disappear when you disconnect from the database, or when you use the DROP VARIABLE statement. Variables are not visible to other connections. Variables are not affected by COMMIT or ROLLBACK statements.

In Version 12.5 and above, variables created with the CREATE VARIABLE statement persist for a connection even when the statement is issued within a (BEGIN...END) statement. You must use DECLARE to create variables that only persist within a (BEGIN...END) statement, for example, within stored procedures.

Variables are useful for creating large text or binary objects for INSERT or UPDATE statements from Embedded SQL programs.

Local variables in procedures and triggers are declared within a compound statement . See “Using compound statements” in Chapter 8, “Using Procedures and Batches” in the *Sybase IQ System Administration Guide*.

Side effects

None.

Standards

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

Permissions

None.

See also

BEGIN... END statement on page 422

DECLARE statement on page 515

DROP VARIABLE statement on page 540

SET statement, Chapter 4, “SQL Data Types”

CREATE VIEW statement

Description

Creates a view on the database. Views are used to give a different perspective on the data even though it is not stored that way.

Syntax

CREATE VIEW
... [*owner*.]*view-name* [(*column-name* [, ...])]
... **AS** *select-without-order-by*
... [**WITH CHECK OPTION**]

Examples

- Creates a view showing all information for male employees only. This view has the same column names as the base table.

```
CREATE VIEW male_employee
AS SELECT *
FROM Employee
WHERE Sex = 'M'
```

- Creates a view showing employees and the departments they belong to:

```
CREATE VIEW emp_dept
AS SELECT emp_lname, emp_fname, dept_name
FROM Employee JOIN Department
ON Employee.dept_id = Department.dept_id
```

Usage

A view can be created for another user by specifying the owner. You must have DBA authority to create a view for another user.

A view name can be used in place of a table name in SELECT, DELETE, UPDATE, and INSERT statements. Views, however, do not physically exist in the database as tables. They are derived each time they are used. The view is derived as the result of the SELECT statement specified in the CREATE VIEW statement. Table names used in a view should be qualified by the user ID of the table owner. Otherwise, a different user ID might not be able to find the table or might get the wrong table.

The columns in the view are given the names specified in the column name list. If the column name list is not specified, then the view columns are given names from the select list items. To use the names from the select list items, the items must be a simple column name or they must have an alias name specified (see SELECT statement on page 632). You cannot add or drop IDENTIY/ AUTOINCREMENT columns from a view.

Views can be updated unless the SELECT statement defining the view contains a GROUP BY clause, an aggregate function, or involves a UNION operation. An update to the view causes the underlying tables to be updated.

view-name An identifier. The default owner is the current user ID.

column-name The columns in the view are given the names specified in the *column-name* list. If the column name list is not specified, the view columns are given names from the select list items. To use the names from the select list items, each item must be a simple column name or have an alias name specified (see SELECT statement on page 632).

AS The SELECT statement on which the view is based must not have an ORDER BY clause on it. It may have a GROUP BY clause and may be a UNION.

WITH CHECK OPTION Rejects any updates and inserts to the view that do not meet the criteria of the views as defined by its SELECT statement. However, Sybase IQ currently ignores this option (it supports the syntax for compatibility reasons).

Side effects

Automatic commit.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Adaptive Server Enterprise.

Permissions

Must have RESOURCE authority and SELECT permission on the tables in the view definition.

See also

CREATE TABLE statement on page 499

DROP statement on page 533

"Copy Definition utility (defncopy)" in Chapter 3, "Database Administration Utilities" in the *Sybase IQ Utility Guide*

DEALLOCATE DESCRIPTOR statement [ESQL]

Description
Frees memory associated with a SQL descriptor area.

Syntax
DEALLOCATE DESCRIPTOR *descriptor-name*:
string

Examples
See ALLOCATE DESCRIPTOR statement [ESQL] on page 394.

Usage
Frees all memory associated with a descriptor area, including the data items, indicator variables, and the structure itself.

Side effects

None.

Standards
- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions
None.

See also
SET DESCRIPTOR statement [ESQL] on page 646

"The Embedded SQL Interface" in *Adaptive Server Anywhere Programming Interface Guide*

Declaration section [ESQL]

Description
Declares host variables in an Embedded SQL program. Host variables are used to exchange data with the database.

Syntax
EXEC SQL BEGIN DECLARE SECTION;
... C declarations
EXEC SQL END DECLARE SECTION;

Examples

```
EXEC SQL BEGIN DECLARE SECTION;
char *emp_lname, initials[5];
int dept;
EXEC SQL END DECLARE SECTION;
```

Usage

A declaration section is simply a section of C variable declarations surrounded by the BEGIN DECLARE SECTION and END DECLARE SECTION statements. A declaration section makes the SQL preprocessor aware of C variables that are used as host variables. Not all C declarations are valid inside a declaration section. See the chapter "Embedded SQL Programming" in the *Adaptive Server Anywhere Programming Guide* for more information.

Standards

- **SQL92**
- **Sybase**

Permissions

None.

See also

BEGIN... END statement on page 422

DECLARE statement

Description

Declares a SQL variable within a compound statement (BEGIN... END).

Syntax

DECLARE *variable_name data-type*

Examples

The following batch illustrates the use of the DECLARE statement and prints a message on the server window:

```
BEGIN
  DECLARE varname CHAR(61) ;
  SET varname = 'Test name';
  MESSAGE name;
END
```

Usage

Variables used in the body of a procedure can be declared using the DECLARE statement. The variable persists for the duration of the compound statement in which it is declared.

The body of a procedure is a compound statement, and variables must be declared immediately following BEGIN. In a Transact-SQL procedure or trigger, there is no such restriction.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** Supported by Adaptive Server Enterprise.
  - To be compatible with Adaptive Server Enterprise, the variable name must be preceded by an @.

- In Adaptive Server Enterprise, a variable that is declared in a procedure or trigger exists for the duration of the procedure or trigger. In Sybase IQ, if a variable is declared inside a compound statement, it exists only for the duration of that compound statement (whether it is declared in a Sybase IQ SQL or Transact-SQL compound statement).

Permissions: None

DECLARE CURSOR statement [ESQL] [SP]

Description: Declares a cursor. Cursors are the primary means for manipulating the results of queries.

Syntax:

DECLARE *cursor-name*
[**SCROLL**
| **NO SCROLL**
| **DYNAMIC SCROLL**
]
CURSOR FOR
{ *select-statement*
| *statement-name*
[**FOR** {**READ ONLY** | **UPDATE** [**OF** *column-name-list*] }]
| **USING** *variable-name* }

Parameters:

cursor-name:
identifier

statement-name:
identifier | host-variable

column-name-list:
identifiers

variable-name:
identifier

Examples:

- Illustrates how to declare a scroll cursor in Embedded SQL:

```
EXEC SQL DECLARE cur_employee SCROLL CURSOR
FOR SELECT * FROM employee ;
```

- Illustrates how to declare a cursor for a prepared statement in Embedded SQL:

```
EXEC SQL PREPARE employee_statement
FROM 'SELECT emp_lname FROM employee' ;
```

```
EXEC SQL DECLARE cur_employee CURSOR
FOR employee_statement ;
```

- Illustrates the use of cursors in a stored procedure:

```
BEGIN
  DECLARE cur_employee CURSOR FOR
      SELECT emp_lname
      FROM employee ;
  DECLARE name CHAR(40) ;
  OPEN cur_employee;
  LOOP
    FETCH NEXT cur_employee INTO name ;
    ...
  END LOOP
  CLOSE cur_employee;
END
```

Usage

The DECLARE CURSOR statement declares a cursor with the specified name for a SELECT statement or a CALL statement.

SCROLL A cursor declared as SCROLL supports the NEXT, PRIOR, FIRST, LAST, ABSOLUTE, and RELATIVE options of the FETCH statement. A SCROLL cursor lets you fetch an arbitrary row in the result set while the cursor is open.

NO SCROLL A cursor declared as NO SCROLL is restricted to moving forward through the result set using only the FETCH NEXT and FETCH ABSOLUTE (0) seek operations.

Since rows cannot be returned to once the cursor leaves the row, there are no sensitivity restrictions on the cursor. Consequently, when a NO SCROLL cursor is requested, Sybase IQ supplies the most efficient kind of cursor, which is an asensitive cursor.

DYNAMIC SCROLL A cursor declared as DYNAMIC SCROLL supports the NEXT, PRIOR, FIRST, LAST, ABSOLUTE, and RELATIVE options of the FETCH statement. A DYNAMIC SCROLL cursor lets you fetch an arbitrary row in the result set while the cursor is open.

FOR statement-name Statements are named using the PREPARE statement. Cursors can be declared only for a prepared SELECT or CALL.

FOR READ ONLY A cursor declared FOR READ ONLY may not be used in a positioned UPDATE or a positioned DELETE operation.

FOR UPDATE You can update the cursor result set of a cursor declared FOR UPDATE. Only asensitive behavior is supported for updatable cursors; any other sensitivity is ignored.

When the cursor is opened, exclusive table locks are taken on all tables that are opened for update. Standalone LOAD TABLE, UPDATE, INSERT, DELETE, and TRUNCATE statements are not allowed on tables that are opened for update in the same transaction, since Sybase IQ permits only one statement to modify a table at a time. You can open only one updatable cursor on a specific table at a time.

Updatable cursors are allowed to scroll, except over Open Client.

READ ONLY is the default value of the FOR clause.

OF column-name-list The list of columns from the cursor result set (specified by the *select-statement*) defined as updatable.

USING variable-name You can declare a cursor on a variable in stored procedures and user-defined functions. The variable is a string containing a SELECT statement for the cursor. The variable must be available when the DECLARE is processed, and so must be one of the following:

- A parameter to the procedure. For example:

```
create function get_row_count(in qry varchar)
returns int
begin
    declare crsr cursor using qry;
    declare rowcnt int;

    set rowcnt = 0;
    open crsr;
    lp: loop
        fetch crsr;
        if SQLCODE <> 0 then leave lp end if;
        set rowcnt = rowcnt + 1;
    end loop;
    return rowcnt;
end
```

- Nested inside another BEGIN…END after the variable has been assigned a value. For example:

```
create procedure get_table_name(
in id_value int, out tabname char(128)
)
begin
declare qry varchar;

set qry = 'select table_name from SYS.SYSTABLE ' ||
 'where table_id=' || string(id_value);
begin
declare crsr cursor using qry;

open crsr;
fetch crsr into tabname;
close crsr;
end
end
```

Embedded SQL

Statements are named using the PREPARE statement. Cursors can be declared only for a prepared SELECT or CALL.

Updatable cursor support

Sybase IQ support of updatable cursors is similar to Adaptive Server Anywhere support of updatable cursors. For a full discussion of cursor types and working with cursors, see the *Adaptive Server Anywhere Programming Guide*. This section contains information important to the use of updatable cursors in Sybase IQ.

Sybase IQ supports one type of cursor sensitivity, which is defined in terms of which changes to underlying data are visible. All Sybase IQ cursors are asensitive, which means that changes might be reflected in the membership, order, or values of the result set seen through the cursor, or might not be reflected at all.

With an asensitive cursor, changes effected by positioned UPDATE and positioned DELETE statements are visible in the cursor result set, except where client-side caching prevents seeing these changes. Inserted rows are not visible.

Rows that are updated so that they no longer meet the requirements of the WHERE clause of the open cursor are still visible.

When using cursors, there is always a tradeoff between efficiency and consistency. Asensitive cursors provide efficient performance at the expense of consistency.

Sybase IQ supports updatable cursors on single tables.

Supported query specifications for updatable cursors in Sybase IQ are as follows:

- Expressions in the select list against columns that are not functionally dependent on columns being updated
- Arbitrary subqueries with asensitive behavior, that is, changes to data referenced by subqueries are not visible in the cursor result set
- ORDER BY clause; the ORDER BY columns may be updated, but the result set does not reorder
- Columns that meet these requirements:
  - No CAST on a column
  - Base columns of a base table in the SELECT clause
  - There are no expressions or functions on that column in the SELECT clause and it is not duplicated in the select list (for example, SELECT c1, c1).
  - Base columns of a base table restricted to those listed in the FOR UPDATE OF *column-name-list* clause, if the clause is specified.

Sybase IQ does *not* permit updatable cursors on queries that contain any operator that precludes a one-to-one mapping of result set rows to rows in a base table; specifically:

- SELECT DISTINCT
- Operator that has a UNION
- Operator that has a GROUP BY
- Operator that has a SET function (single group or extended GROUP BY)
- Operator that has an OLAP function, with the exception of RANK()

See the description of the UPDATE (positioned) statement [ESQL] [SP] on page 664 for information on the columns and expressions allowed in the SET clause for the update of a row in the result set of a cursor.

Sybase IQ supports inserts only on updatable cursors where all nonnullable, nonidentity columns are both selected and updatable.

In Sybase IQ, COMMIT and ROLLBACK are not allowed inside an open updatable cursor, even if the cursor is opened as a hold cursor. Sybase IQ does support ROLLBACK TO SAVEPOINT inside an updatable cursor.

Any failure that occurs after the cursor is open results in a rollback of all operations that have been performed through this open cursor.

Updatable cursor limitations

A declared cursor is read-only and not updatable in cases where:

- The data extraction facility is enabled with the TEMP_EXTRACT_NAME1 option set to a pathname
- As a join index, or within a join index
- ANSI_CLOSE_CURSORS_ON_ROLLBACK is set OFF
- CHAINED is set OFF
- The statement is INSERT SELECT or SELECT INTO
- More than one table is included
- No updatable columns exist

If Sybase IQ fails to set an updatable cursor when requested, see the *.iqmsg* file for related information.

There is a limitation regarding updatable cursors and ODBC. A maximum of 65535 rows or records can be updated, deleted, or inserted at a time using the following ODBC functions:

- SQLSetPos SQL_UPDATE, SQL_DELETE, and SQL_ADD
- SQLBulkOperations SQL_ADD, SQL_UPDATE_BY_BOOKMARK, and SQL_DELETE_BY_BOOKMARK

There is an implementation-specific limitation to the maximum value in the statement attribute that controls the number of effected rows to the largest value of an UNSIGNED SMALL INT, which is 65535.

```
SQLSetStmtAttr(HANDLE,SQL_ATTR_ROW_ARRAY_SIZE,
VALUE,0)
```

This information should be added to “Updatable cursor limitations” in the “Usage” section for the DECLARE CURSOR statement description in the “SQL Statements” chapter.

Updatable cursor differences

Sybase IQ updatable cursors differ from ANSI SQL3 standard behavior as follows:

- Hold cursor update close on commit.
- Sybase IQ locks tables when the cursor is open.

- All updates, deletes, and insert operations are applied when the cursor is closed, in the following order: deletes first, then updates, then inserts.

Side effects

None.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions

None.

See also

CALL statement on page 429

DELETE (positioned) statement [ESQL] [SP] on page 527

OPEN statement [ESQL] [SP] on page 603

PREPARE statement [ESQL] on page 611

SELECT statement on page 632

UPDATE (positioned) statement [ESQL] [SP] on page 664

“sp_iqcursorinfo procedure” on page 759

DECLARE CURSOR statement [T-SQL]

Description

Declares a cursor in a manner compatible with Adaptive Server Enterprise.

Syntax

DECLARE *cursor-name*
... **CURSOR FOR** *select-statement*
... [**FOR** { **READ ONLY** | **UPDATE** }]

Usage

Sybase IQ supports a DECLARE CURSOR syntax that is not supported in Adaptive Server Enterprise. For information on the full DECLARE CURSOR syntax, see DECLARE CURSOR statement [ESQL] [SP] on page 516.

This section describes the overlap between the Sybase IQ and Adaptive Server Enterprise versions of DECLARE CURSOR.

Side effects

None.

Standards

- **SQL92** Entry-level compliant. The FOR UPDATE and FOR READ ONLY options are Transact-SQL extensions.

- **Sybase** There are some features of the Adaptive Server Enterprise DECLARE CURSOR statement that are not supported in Sybase IQ.
  - In the Sybase IQ dialect, DECLARE CURSOR in a procedure or batch must immediately follow the BEGIN keyword. In the Transact-SQL dialect, there is no such restriction.
  - In Adaptive Server Enterprise, when a cursor is declared in a procedure or batch, it exists for the duration of the procedure or batch. In Sybase IQ, if a cursor is declared inside a compound statement, it exists only for the duration of that compound statement (whether it is declared in a Sybase IQ or Transact-SQL compound statement).

Permissions: None.

See also: DECLARE CURSOR statement [ESQL] [SP] on page 516

"sp_iqcursorinfo procedure" on page 759

DECLARE LOCAL TEMPORARY TABLE statement

Description: Declares a local temporary table.

Syntax:

DECLARE LOCAL TEMPORARY TABLE *table-name*
... (*column-definition* [*column-constraint*]...
[, *column-definition* [*column-constraint*]...]
[, *table-constraint*]...)
... [**ON COMMIT** { **DELETE** | **PRESERVE** } **ROWS**
NOT TRANSACTIONAL]

Examples:

- Illustrates how to declare a local temporary table in Embedded SQL:

```
EXEC SQL DECLARE LOCAL TEMPORARY TABLE MyTable (
  number INT
    );
```

- Illustrates how to declare a local temporary table in a stored procedure:

```
BEGIN
  DECLARE LOCAL TEMPORARY TABLE TempTab (
    number INT
  );
  ...
END
```

Usage: The DECLARE LOCAL TEMPORARY TABLE statement declares a temporary table.

A local temporary table and the rows in it are visible only to the connection that created the table and inserted the rows. By default, the rows of a temporary table are deleted on COMMIT.

Declared local temporary tables within compound statements exist within the compound statement. Otherwise, the declared local temporary table exists until the end of the connection.

See CREATE TABLE statement on page 499 for definitions of *column-definition*, *column-constraint*, and *table-constraint*, and the NOT TRANSACTIONAL clause. See SELECT statement on page 632 for an example of how to select data into a temporary table.

Once you create a local temporary table, either implicitly or explicitly, you cannot create another temporary table of that name for as long as the temporary table exists. For example, you could create a local temporary table implicitly by entering:

```
select * into #tmp from table1
```

Or, you could create a local temporary table explicitly by declaring it:

```
declare local temporary table foo
```

If you then try to select into #tmp or foo, or declare #tmp or foo again, you receive an error indicating that #tmp or foo already exists.

When you declare a local temporary table, omit the owner specification. If you specify the same owner.table in more than one DECLARE LOCAL TEMPORARY TABLE statement in the same session, a syntax error is reported. For example, an error is reported when the following statements are executed in the same session:

```
DECLARE LOCAL TEMPORARY TABLE user1.temp(col1 int);
DECLARE LOCAL TEMPORARY TABLE user1.temp(col1 int);
```

If the owner name is omitted, then the error “Item temp already exists” is reported:

```
DECLARE LOCAL TEMPORARY TABLE temp(col1 int);
DECLARE LOCAL TEMPORARY TABLE temp(col1 int);
```

You cannot use the ALTER TABLE and DROP INDEX statements on local temporary tables.

You cannot use the sp_iqindex, sp_iqtablesize, and sp_iqindexsize stored procedures on local temporary tables.

Side effects

None.

Standards
- **SQL92** Conforms to SQL92 standard
- **Sybase** Adaptive Server Enterprise does not support DECLARE TEMPORARY TABLE.

Permissions: None.

See also: CREATE TABLE statement on page 499

SELECT statement on page 632

DELETE statement

Description: Deletes rows from the database.

Syntax:
DELETE [**FROM**] [*owner*.]*table-name*
... [**FROM** *table-list*]
... [**WHERE** *search-condition*]

Examples
- Removes employee 105 from the database:

```
DELETE
FROM employee
WHERE emp_id = 105
```

- Removes all data prior to 1993 from the fin_data table:

```
DELETE
FROM fin_data
WHERE year < 1993
```

- Removes all names from the contact table if they are already present in the customer table:

```
DELETE
FROM contact
FROM contact, customer
WHERE contact.last_name = customer.lname
AND contact.first_name = customer.fname
```

Usage

DELETE deletes all the rows from the named table that satisfy the search condition. If no WHERE clause is specified, all rows from the named table are deleted.

DELETE can be used on views provided the SELECT statement defining the view has only one table in the FROM clause and does not contain a GROUP BY clause, an aggregate function, or involve a UNION operation.

The optional second FROM clause in the DELETE statement allows rows to be deleted based on joins. If the second FROM clause is present, the WHERE clause qualifies the rows of this second FROM clause. Rows are deleted from the table name given in the first FROM clause.

The effects of a DELETE on a table can be passed on to any of the join indexes that reference that table through the SYNCHRONIZE JOIN INDEX command. For performance reasons, you should do as many deletes as possible before synchronizing the join indexes.

Note You cannot use the DELETE statement on a join virtual table. If you attempt to delete from a join virtual table, an error is reported.

Correlation name resolution

The following statement illustrates a potential ambiguity in table names in DELETE statements with two FROM clauses that use correlation names:

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_2 AS alias_2
WHERE ...
```

The table table_1 is identified without a correlation name in the first FROM clause, but with a correlation name in the second FROM clause. In this case, table_1 in the first clause is identified with alias_1 in the second clause; there is only one instance of table_1 in this statement.

This is an exception to the general rule that where a table is identified with a correlation name and without a correlation name in the same statement, two instances of the table are considered.

Consider the following example:

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

In this case, there are two instances of table_1 in the second FROM clause. There is no way of identifying which instance the first FROM clause should be identified with. The usual rules of correlation names apply, and table_1 in the first FROM clause is identified with neither instance in the second clause: there are three instances of table_1 in the statement.

Side effects

None.

Standards

- **SQL92** Entry-level compliant. The use of more than one table in the FROM clause is a vendor extension.
- **Sybase** Supported by Adaptive Server Enterprise, including the vendor extension.

  The Transact-SQL ROWCOUNT option has no effect on DELETE operations in Sybase IQ.

Permissions

Must have DELETE permission on the table.

See also

FROM clause on page 553

INSERT statement on page 568

SYNCHRONIZE JOIN INDEX statement on page 657

TRUNCATE TABLE statement on page 658

DELETE (positioned) statement [ESQL] [SP]

Description

Deletes the data at the current location of a cursor.

Syntax

DELETE
WHERE CURRENT OF *cursor-name*

Parameters

cursor-name:
identifier | hostvar

Examples

The following statement removes the current row from the database:

```
DELETE WHERE CURRENT OF cur_employee
```

Usage

This form of the DELETE statement deletes the current row of the specified cursor. The current row is defined to be the last row fetched from the cursor.

The positioned DELETE statement can be used on a cursor open on a view as long as the view is updatable.

Changes effected by positioned DELETE statements are visible in the cursor result set, except where client-side caching prevents seeing these changes.

Standards

- **SQL92** Entry-level feature. The range of cursors that can be updated may contain vendor extensions if the ANSI_UPDATE_CONSTRAINTS option is set to OFF.
- **SQL99** Core feature. The range of cursors that can be updated may contain vendor extensions if the ANSI_UPDATE_CONSTRAINTS option is set to OFF.

- **Sybase** Embedded SQL use is supported by Open Client/Open Server. Procedure and trigger use is supported in Adaptive Server Anywhere.

Permissions

Must have DELETE permission on tables used in the cursor.

See also

DECLARE CURSOR statement [ESQL] [SP] on page 516

INSERT statement on page 568

UPDATE statement on page 661

UPDATE (positioned) statement [ESQL] [SP] on page 664

"sp_iqcursorinfo procedure" on page 759

DESCRIBE statement [ESQL]

Description

Gets information about the host variables required to store data retrieved from the database or host variables used to pass data to the database.

Syntax

DESCRIBE
... [**USER TYPES**]
... [{ **ALL** | **BIND VARIABLES FOR** | **INPUT**
| **OUTPUT** | **SELECT LIST FOR** }]
... [{ **LONG NAMES** [*long-name-spec*] | **WITH VARIABLE RESULT** }]
... [**FOR**] { *statement-name* | **CURSOR** *cursor-name* }
... **INTO** *sqlda-name*

Parameters

long-name-spec:
{ OWNER.TABLE.COLUMN | TABLE.COLUMN | COLUMN }

statement-name:
identifier | host-variable

cursor-name:
declared cursor

sqlda-name:
identifier

Examples

The following example shows how to use the DESCRIBE statement:

```
sqlda = alloc_sqlda( 3 );
EXEC SQL DESCRIBE OUTPUT
  FOR employee_statement
  INTO sqlda;
if( sqlda->sqld  >  sqlda->sqln ) {
  actual_size = sqlda->sqld;
```

```
    free_sqlda( sqlda );
    sqlda = alloc_sqlda( actual_size );
    EXEC SQL DESCRIBE OUTPUT
      FOR employee_statement
      INTO sqlda;
}
```

Usage

DESCRIBE sets up the named SQLDA to describe either the OUTPUT (equivalently SELECT LIST) or the INPUT (BIND VARIABLES) for the named statement.

In the INPUT case, DESCRIBE BIND VARIABLES does not set up the data types in the SQLDA: this needs to be done by the application. The ALL keyword lets you describe INPUT and OUTPUT in one SQLDA.

If you specify a statement name, the statement must have been previously prepared using the PREPARE statement with the same statement name and the SQLDA must have been previously allocated (see the ALLOCATE DESCRIPTOR statement [ESQL] on page 394).

If you specify a cursor name, the cursor must have been previously declared and opened. The default action is to describe the OUTPUT. Only SELECT statements and CALL statements have OUTPUT. A DESCRIBE OUTPUT on any other statement indicates no output by setting the sqld field of the SQLDA to zero.

USER TYPES A DESCRIBE statement with the USER TYPES clause returns information about user-defined data types of a column. Typically, such a DESCRIBE is done when a previous DESCRIBE returns an indicator of DT_HAS_USERTYPE_INFO.

The information returned is the same as for a DESCRIBE without the USER TYPES keywords, except that the sqlname field holds the name of the user-defined data type, instead of the name of the column.

If DESCRIBE uses the LONG NAMES clause, the sqldata field holds this information.

SELECT DESCRIBE OUTPUT fills in the data type and length in the SQLDA for each select list item. The name field is also filled in with a name for the select list item. If an alias is specified for a select list item, the name is that alias. Otherwise, the name derives from the select list item: if the item is a simple column name, it is used; otherwise, a substring of the expression is used. DESCRIBE also puts the number of select list items in the sqld field of the SQLDA.

If the statement being described is a UNION of two or more SELECT statements, the column names returned for DESCRIBE OUTPUT are the same column names which would be returned for the first SELECT statement.

CALL The DESCRIBE OUTPUT statement fills in the data type, length, and name in the SQLDA for each INOUT or OUT parameter in the procedure. DESCRIBE OUTPUT also puts the number of INOUT or OUT parameters in the sqld field of the SQLDA.

CALL (result set) DESCRIBE OUTPUT fills in the data type, length, and name in the SQLDA for each RESULT column in the procedure definition. DESCRIBE OUTPUT also puts the number of result columns in the sqld field of the SQLDA.

A bind variable is a value supplied by the application when the database executes the statements. Bind variables can be considered parameters to the statement. DESCRIBE INPUT fills in the name fields in the SQLDA with the bind variable names. DESCRIBE INPUT also puts the number of bind variables in the sqld field of the SQLDA.

DESCRIBE uses the indicator variables in the SQLDA to provide additional information. DT_PROCEDURE_IN and DT_PROCEDURE_OUT are bits that are set in the indicator variable when a CALL statement is described. DT_PROCEDURE_IN indicates an IN or INOUT parameter and DT_PROCEDURE_OUT indicates an INOUT or OUT parameter. Procedure RESULT columns has both bits clear. After a describe OUTPUT, these bits can be used to distinguish between statements that have result sets (need to use OPEN, FETCH, RESUME, CLOSE) and statements that do not (need to use EXECUTE). DESCRIBE INPUT sets DT_PROCEDURE_IN and DT_PROCEDURE_OUT appropriately only when a bind variable is an argument to a CALL statement; bind variables within an expression that is an argument in a CALL statement sets the bits.

DESCRIBE ALL lets you describe INPUT and OUTPUT with one request to the database server. This has a performance benefit in a multiuser environment. The INPUT information is filled in the SQLDA first, followed by the OUTPUT information. The sqld field contains the total number of INPUT and OUTPUT variables. The DT_DESCRIBE_INPUT bit in the indicator variable is set for INPUT variables and clear for OUTPUT variables.

Retrieving long column names

The LONG NAMES clause is provided to retrieve column names for a statement or cursor. Without this clause, there is a 29-character limit on the length of column names: with the clause, names of an arbitrary length are supported.

If LONG NAMES is used, the long names are placed into the SQLDATA field of the SQLDA, as if you were fetching from a cursor. None of the other fields (SQLLEN, SQLTYPE, and so on) are filled in. The SQLDA must be set up like a FETCH SQLDA: it must contain one entry for each column, and the entry must be a string type.

The default specification for the long names is TABLE.COLUMN.

Describing variable result sets

The WITH VARIABLE RESULT statement is used to describe procedures that might have more than one result set, with different numbers or types of columns.

If WITH VARIABLE RESULT is used, the database server sets the SQLCOUNT value after the describe to one of the following values:

- **0** The result set may change: the procedure call should be described again following each OPEN statement.
- **1** The result set is fixed. No re-describing is required.

For more information on the use of the SQLDA structure, see the chapter "Embedded SQL Programming" in the *Adaptive Server Anywhere Programming Guide*.

Side effects

None.

Standards

- **SQL92** Part of the SQL92 standard. Some clauses are vendor extensions.
- **Sybase** Some clauses supported by Open Client/Open Server.

Permissions

None.

See also

DECLARE CURSOR statement [ESQL] [SP] on page 516

OPEN statement [ESQL] [SP] on page 603

PREPARE statement [ESQL] on page 611

DISCONNECT statement [DBISQL]

Description

Drops a connection with the database.

Syntax

DISCONNECT [{ *connection-name* | **CURRENT** | **ALL** }]

Parameters

connection-name:
identifier, string, or host-variable

Examples

- The following statement shows how to use DISCONNECT in Embedded SQL:

```
EXEC SQL DISCONNECT :conn_name
```

- The following statement shows how to use DISCONNECT from DBISQL to disconnect all connections:

```
DISCONNECT ALL
```

Usage

The DISCONNECT statement drops a connection with the database server and releases all resources used by it. If the connection to be dropped was named on the CONNECT statement, then the name can be specified. Specifying ALL drops all of the application's connections to all database environments. CURRENT is the default and drops the current connection.

An implicit ROLLBACK is executed on connections that are dropped.

Side effects

None.

Standards

- **SQL92** Intermediate-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions

None

See also

CONNECT statement [ESQL] [DBISQL] on page 439

SET CONNECTION statement [DBISQL] [ESQL] on page 645

DROP statement

Description

Removes objects from the database.

Syntax

DROP
{ **DBSPACE** *dbspace-name*
| { **DATATYPE** | **DOMAIN** } *datatype-name*
| **EVENT** *event-name*
| **INDEX** [[*owner*].*table-name*.]*index-name*
| **JOIN INDEX** [*owner*.]*join-index-name*
| **MESSAGE** *message-number*
| **TABLE** [*owner*.]*table-name*
| **VIEW** [*owner*.]*view-name*
| **PROCEDURE** [*owner*.]*procedure-name*
| **FUNCTION** [*owner*.]*function-name* }

Examples

- Drops the department table from the database:

```
DROP TABLE department
```

- Drops the emp_dept view from the database:

```
DROP VIEW emp_dept
```

Usage

DROP removes the definition of the indicated database structure. If the structure is a dbspace, then all tables with any data in that dbspace must be dropped or relocated prior to dropping the dbspace; other structures are automatically relocated. If the structure is a table, all data in the table is automatically deleted as part of the dropping process. Also, all indexes and keys for the table are dropped by DROP TABLE. However, you cannot drop the table if any join indexes use that table. You must first use DROP JOIN INDEX to remove the join indexes.

Note In a Sybase IQ multiplex, always perform DROP JOIN INDEX operations in single-node mode on the write server, then synchronize query servers. DROP JOIN INDEX returns an error instead of propagating from write server to query server.

DROP INDEX deletes any explicitly created index. It only deletes an implicitly created index if there is no associated primary key, unique, or foreign-key constraints.

DROP INDEX for a nonunique HG index fails if an associated unenforced foreign key exists.

Warning! Do not delete views owned by the DBO user. Deleting such views or changing them into tables might cause problems.

DROP TABLE, DROP INDEX, DROP JOIN INDEX, and DROP DBSPACE are prevented whenever the statement affects a table that is currently being used by another connection.

DROP TABLE is prevented if the primary table has foreign-key constraints associated with it, including unenforced foreign-key constraints

DROP TABLE is also prevented if the table has an IDENTITY column and IDENTITY_INSERT is set to that table. To drop the table you must clear IDENTITY_INSERT, that is, set it to ' ' (an empty string), or set it to another table name.

A foreign key can have either a nonunique single or a multicolumn HG index. A primary key may have unique single or multicolumn HG indexes. You cannot drop the HG index implicitly created for an existing foreign key, primary key, and unique constraint. If a DBA is dropping a join index belonging to another user, the join index name must be qualified with an owner name.

The four initial dbspaces are SYSTEM, IQ_SYSTEM_MAIN, IQ_SYSTEM_TEMP, and IQ_SYSTEM_MSG. Any dbspace, except SYSTEM and IQ_SYSTEM_MSG, can be dropped using DROP DBSPACE, as long as there is at least one remaining dbspace with readwrite mode. You must relocate or drop tables in the dbspace, before you can drop the dbspace. An error is returned if the dbspace still contains user data; other structures are automatically relocated when the dbspace is dropped. Dbspace names are case sensitive for databases created with CASE RESPECT.

Note A dbspace may contain data at any point after it is used by a command, thereby preventing a DROP DBSPACE on it.

See the section “Working with dbspaces” in Chapter 5, “Working with Database Objects” in the *Sybase IQ System Administration Guide* for more information on modifying dbspaces.

DROP PROCEDURE is prevented when the procedure is in use by another connection.

DROP DATATYPE is prevented if the data type is used in a table. You must change data types on all columns defined on the user-defined data type to drop the data type. It is recommended that you use DROP DOMAIN rather than DROP DATATYPE, as DROP DOMAIN is the syntax used in the ANSI/ISO SQL3 draft.

Note Do not use DROP DOMAIN on a multiplex query server without a local IQ Main Store. Synchronizing the multiplex removes domains from query servers without local stores. If the Query Server has a local store, then both CREATE DOMAIN and DROP DOMAIN are permitted.

Side effects

Automatic commit. Clears the Data window in DBISQL. DROP TABLE and DROP INDEX close all cursors for the current connection.

Local temporary tables are an exception; no commit is performed when one is dropped.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Adaptive Server Enterprise.

Permissions

For DROP DBSPACE, must have DBA authority and must be the only connection to the database.

For others, must be the owner of the object, or have DBA authority.

Global temporary tables cannot be dropped unless all users that have referenced the temporary table have disconnected.

See also

ALTER DBSPACE statement on page 398

ALTER TABLE statement on page 409

CREATE DBSPACE statement on page 453

CREATE DOMAIN statement on page 456

CREATE EVENT statement on page 458

CREATE INDEX statement on page 473

CREATE MESSAGE statement [T-SQL] on page 484

CREATE PROCEDURE statement on page 485

CREATE TABLE statement on page 499

CREATE VIEW statement on page 512

“sp_iqdbspace procedure” in Chapter 10, “System Procedures”

Chapter 5, “Working with Database Objects” in the *Sybase IQ System Administration Guide*.

DROP CONNECTION statement

Description
Drops any user’s connection to the database.

Syntax
DROP CONNECTION *connection-id*

Examples
- The following statement drops connection with ID number 4:

```
DROP CONNECTION 4
```

Usage
DROP CONNECTION disconnects a user from the database by dropping the connection to the database.

The *connection-id* for the connection is obtained using the connection_property function to request the connection number. The following statement returns the connection ID of the current connection:

```
SELECT connection_property( 'number' )
```

Side effects

None.

Standards
- **SQL92** Vendor extension.
- **Sybase** Not supported by Adaptive Server Enterprise.

Permissions
Must have DBA authority.

See also
CONNECT statement [ESQL] [DBISQL] on page 439

DROP DATABASE statement

Description
Drops a database and its associated dbspace segment files.

Syntax
DROP DATABASE *db-filename* [**KEY** *key-spec*]

Parameters
key-spec:
A string, including mixed cases, numbers, letters, and special characters. It might be necessary to protect the key from interpretation or alteration by the command shell.

Examples

- Drops database mydb:

```
DROP DATABASE mydb.db
```

- Drops the encrypted database *marvin.db*, which was created with the key *is!seCret*:

```
DROP DATABASE 'marvin.db' KEY 'is!seCret'
```

- The following UNIX example drops the database temp.db from the */s1/temp* directory:

```
DROP DATABASE '/s1/temp/temp.db'
```

Usage

DROP DATABASE drops all the database segment files associated with the IQ Store and Temporary Store before it drops the Catalog store files.

The database must be stopped before you can drop it. If the connection parameter AUTOSTOP=no is used, you might need to issue a STOP DATABASE statement.

The *db-filename* you specify corresponds to the database filename you defined for the database using CREATE DATABASE. If you specified a directory path for this value in the CREATE DATABASE command, *you must also specify the directory path* for DROP DATABASE. Otherwise, Sybase IQ looks for the database files in the default directory where the server files reside.

If you use Interactive SQL instead of Sybase Central to drop databases, always provide an explicit path. For example, if you drop the write server's database before dropping the query servers on the same machine, the following might return an error:

```
DROP DATABASE 'mydbname'
```

To avoid the error, specify the full database path, for example:

```
DROP DATABASE '/s1/mpx/wsrvr/mydbname.db'
```

You cannot execute a DROP DATABASE statement to drop an IQ database that has a DatabaseStart event defined for it.

Side effects

None.

Standards

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

Permissions

Required permissions are set using the database server -gu command line option. The default setting is to require DBA authority.

See also

CREATE DATABASE statement on page 442

DROP EXTERNLOGIN statement

Description
Drops an external login from the Sybase IQ system tables.

Syntax
DROP EXTERNLOGIN *login-name*
TO *remote-server*

Examples
```
DROP EXTERNLOGIN dba TO sybase1
```

Usage
DROP EXTERNLOGIN deletes an external login from the Sybase IQ system tables.

login-name Specifies the local user login name.

TO The TO clause specifies the name of the remote server. The local user's alternate login name and password for that server is the external login that is deleted.

Side effects

Automatic commit.

Standards
- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions
Only the login name and the DBA account can delete an external login for login name.

See also
CREATE EXTERNLOGIN statement on page 467

DROP SERVER statement

Description
Drops a remote server from the Sybase IQ system tables.

Syntax
DROP SERVER *server-name*

Examples
```
DROP SERVER ase_prod
```

Usage
You must drop all the proxy tables that have been defined for the remote server before this statement succeeds.

Side effects

Automatic commit.

Standards
- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions
Only the DBA account can delete a remote server.

See also CREATE SERVER statement on page 494

DROP SERVICE statement

Description Deletes a Web service.

Syntax **DROP SERVICE** *service-name*

Examples To drop a Web service named “tables”, execute the following statement:

```
DROP SERVICE tables
```

Usage DROP SERVICE deletes a Web service.

Side effects

None.

Standards
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise.

Permissions Must have DBA authority.

See also ALTER SERVICE statement on page 407

CREATE SERVICE statement on page 496

“Using the Built-in Web Server” in *Adaptive Server Anywhere Database Administration Guide*

DROP STATEMENT statement [ESQL]

Description Frees statement resources.

Syntax **DROP STATEMENT** [*owner*.]*statement-name*

Parameters *statement-name*:
identifier or host-variable

Examples The following are examples of DROP STATEMENT use:

```
EXEC SQL DROP STATEMENT S1;
EXEC SQL DROP STATEMENT :stmt;
```

Usage

DROP STATEMENT frees resources used by the named prepared statement. These resources are allocated by a successful PREPARE statement, and are normally not freed until the database connection is released.

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not supported in Open Client/Open Server

Permissions

Must have prepared the statement.

See also

PREPARE statement [ESQL] on page 611

DROP VARIABLE statement

Description

Eliminates a SQL variable.

Syntax

DROP VARIABLE *identifier*

Usage

DROP VARIABLE eliminates a SQL variable previously created using CREATE VARIABLE. Variables are automatically eliminated when the database connection is released. However, variables are often used for large objects. Thus, eliminating them after use might free up significant resources (primarily disk space).

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not supported in Adaptive Server Enterprise.

Permissions

None

See also

CREATE VARIABLE statement on page 511

SET statement on page 641

EXECUTE statement [ESQL]

Description

Executes a SQL statement.

Syntax

Syntax 1

EXECUTE *statement-name*
... [{ **USING DESCRIPTOR** *sqlda-name* | **USING** *host-variable-list* }]
... [{ **INTO DESCRIPTOR** *into-sqlda-name* | **INTO** *into-host-variable-list*]
... [**ARRAY** :*nnn* }]

Syntax 2

EXECUTE IMMEDIATE *statement*

Parameters

statement-name:
identifier or host-variable

sqlda-name:
identifier

into-sqlda-name:
identifier

statement:
string or host-variable

Examples

- Executes a DELETE:

```
EXEC SQL EXECUTE IMMEDIATE
'DELETE FROM employee WHERE emp_id = 105';
```

- Executes a prepared DELETE statement:

```
EXEC SQL PREPARE del_stmt FROM
'DELETE FROM employee WHERE emp_id = :a';
EXEC SQL EXECUTE del_stmt USING :employee_number;
```

- Executes a prepared query:

```
EXEC SQL PREPARE sel1 FROM
'SELECT emp_lname FROM employee WHERE emp_id = :a';
EXEC SQL EXECUTE sel1 USING :employee_number INTO
:emp_lname;
```

Usage

Syntax 1 executes the named dynamic statement that was previously prepared. If the dynamic statement contains host variable placeholders which supply information for the request (bind variables), then either the sqlda-name must specify a C variable which is a pointer to an SQLDA containing enough descriptors for all bind variables occurring in the statement, or the bind variables must be supplied in the host-variable-list.

The optional ARRAY clause can be used with prepared INSERT statements, to allow wide inserts, which insert more than one row at a time and which might improve performance. The value nnn is the number of rows to be inserted. The SQLDA must contain nnn * (columns per row) variables. The first row is placed in SQLDA variables 0 to (columns per row)-1, and so on.

OUTPUT from a SELECT statement or a CALL statement is put either into the variables in the variable list or into the program data areas described by the named SQLDA. The correspondence is one to one from the OUTPUT (selection list or parameters) to either the host variable list or the SQLDA descriptor array.

If EXECUTE is used with an INSERT statement, the inserted row is returned in the second descriptor. For example, when using autoincrement primary keys that generate primary-key values, EXECUTE provides a mechanism to refetch the row immediately and determine the primary-key value assigned to the row.

Syntax 2 is a short form to PREPARE and EXECUTE a statement that does not contain bind variables or output. The SQL statement contained in the string or host variable is immediately executed.

EXECUTE can be used for any SQL statement that can be prepared. Cursors are used for SELECT statements or CALL statements that return many rows from the database.

After successful execution of an INSERT, UPDATE, or DELETE statement, the *sqlerrd[2]* field of the SQLCA (SQLCOUNT) is filled in with the number of rows affected by the operation.

Side effects

None.

Standards

- **SQL92** Intermediate-level feature.
- **Sybase** Supported in Open Client/Open Server.

Permissions

Permissions are checked on the statement being executed.

See also

DECLARE CURSOR statement [ESQL] [SP] on page 516

PREPARE statement [ESQL] on page 611

EXECUTE statement [T-SQL]

Description

Invokes a procedure, as an Adaptive Server Enterprise-compatible alternative to the CALL statement.

Syntax

EXECUTE [*@return_status* =] [*owner*.]*procedure_name*
... { [*@parameter-name* =] *expression*
| [*@parameter-name* =] *@variable* [*output*] } ,...

Examples

- Illustrates the EXECUTE statement:

```
CREATE PROCEDURE p1( @var INTEGER = 54 )
AS
PRINT 'on input @var = %1! ', @var
DECLARE @internal_var integer
SELECT @intvar=123
SELECT @var=@intvar
PRINT 'on exit @var = %1!', @var
```

- The following statement executes the procedure, supplying the input value of 23 for the parameter. If you are connected from an Open Client application, PRINT messages are displayed on the client window. If you are connected from an ODBC or Embedded SQL application, messages display on the database server window.

```
EXECUTE p1 23
```

- An alternative way of executing the procedure, which is useful if there are several parameters:

```
EXECUTE p1 @var = 23
```

- Executes the procedure, using the default value for the parameter:

```
EXECUTE p1
```

- Executes the procedure, and stores the return value in a variable for checking return status:

```
EXECUTE @status = p1 23
```

Usage

EXECUTE executes a stored procedure, optionally supplying procedure parameters and retrieving output values and return status information.

EXECUTE is implemented for Transact-SQL compatibility, but can be used in either Transact-SQL or Sybase IQ batches and procedures.

Side effects

None.

Permissions — Must be the owner of the procedure, have EXECUTE permission for the procedure, or have DBA authority.

See also — CALL statement on page 429

EXECUTE IMMEDIATE statement [ESQL] [SP]

Description — Enables dynamically constructed statements to be executed from within a procedure.

Syntax

Syntax 1

EXECUTE IMMEDIATE [*execute-option*] *string-expression*

execute-option:
WITH QUOTES [**ON** | **OFF**]
| **WITH ESCAPES** { **ON** | **OFF** }
| **WITH RESULT SET** { **ON** | **OFF** }

Syntax 2

EXECUTE (*string-expression*)

Examples — The following procedure creates a table, where the table name is supplied as a parameter to the procedure. The full EXECUTE IMMEDIATE statement must be on a single line.

```
CREATE PROCEDURE CreateTableProc(
                    IN tablename char(30)
                    )
BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE ' || tablename ||
' ( column1 INT PRIMARY KEY)'
END
```

To call the procedure and create a table mytable:

```
CALL CreateTableProc( 'mytable' )
```

Usage — EXECUTE IMMEDIATE extends the range of statements that can be executed from within procedures. It lets you execute dynamically prepared statements, such as statements that are constructed using the parameters passed in to a procedure.

Literal strings in the statement must be enclosed in single quotes, and must differ from any existing statement name in a PREPARE or EXECUTE IMMEDIATE statement. The statement must be on a single line.

Only global variables can be referenced in a statement executed by EXECUTE IMMEDIATE.

Only syntax 2 can be used inside Transact-SQL stored procedures.

WITH QUOTES When you specify WITH QUOTES or WITH QUOTES ON, any double quotes in the string expression are assumed to delimit an identifier. When you do not specify WITH QUOTES, or specify WITH QUOTES OFF, the treatment of double quotes in the string expression depends on the current setting of the QUOTED_IDENTIFIER option.

WITH QUOTES is useful when an object name that is passed into the stored procedure is used to construct the statement that is to be executed, but the name might require double quotes and the procedure might be called when QUOTED_IDENTIFIER is set to OFF.

For more information, see "QUOTED_IDENTIFIER option [TSQL]" on page 143

WITH ESCAPES WITH ESCAPES OFF causes any escape sequences (such as \n, \x, or \\) in the string expression to be ignored. For example, two consecutive backslashes remain as two backslashes, rather than being converted to a single backslash. The default setting is equivalent to WITH ESCAPES ON.

You can use WITH ESCAPES OFF for easier execution of dynamically constructed statements referencing file names that contain backslashes.

In some contexts, escape sequences in the *string-expression* are transformed before EXECUTE IMMEDIATE is executed. For example, compound statements are parsed before being executed, and escape sequences are transformed during this parsing, regardless of the WITH ESCAPES setting. In these contexts, WITH ESCAPES OFF prevents further translations from occurring. For example:

```
BEGIN
DECLARE String1 LONG VARCHAR;
DECLARE String2 LONG VARCHAR;
EXECUTE IMMEDIATE
  'SET String1 = ''One backslash: \\\\ ''';
 EXECUTE IMMEDIATE WITH ESCAPES OFF
  'SET String2 = ''Two backslashes: \\\\ ''';
 SELECT String1, String2
END
```

WITH RESULT SET You can have an EXECUTE IMMEDIATE statement return a result set by specifying WITH RESULT SET ON. With this clause, the containing procedure is marked as returning a result set. If you do not include this clause, an error is reported when the procedure is called if the statement does not produce a result set.

Note The default option is WITH RESULT SET OFF, meaning that no result set is produced when the statement is executed.

Side effects

None. However, if the statement is a data definition statement with an automatic commit as a side effect, then that commit does take place.

Standards

- **SQL92** Intermediate-level feature.
- **Sybase** Supported in Open Client/Open Server.

Permissions

None. The statement is executed with the permissions of the owner of the procedure, not with the permissions of the user who calls the procedure.

See also

BEGIN... END statement on page 422

CREATE PROCEDURE statement on page 485

EXIT statement [DBISQL]

Description

Leaves DBISQL.

Syntax

{ **EXIT** | **QUIT** | **BYE** }

Usage

Leaves the DBISQL environment and return to the operating system. This closes your connection with the database. Before doing so, DBISQL performs a COMMIT if the COMMIT_ON_EXIT option is ON. If the option is OFF, DBISQL performs a ROLLBACK. The default action is to COMMIT any changes you have made to the database.

Side effects

Does a commit if option COMMIT_ON_EXIT is ON (default); otherwise does do a rollback.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not applicable in Adaptive Server Enterprise.

Permissions None

See also SET OPTION statement on page 647

FETCH statement [ESQL] [SP]

Description Repositions a cursor and gets data from it.

Syntax
FETCH
{ **NEXT** | **PRIOR** | **FIRST** | **LAST**
| **ABSOLUTE** *row-count* | **RELATIVE** *row-count* }
... *cursor-name*
... { [**INTO** *host-variable-list*]
| **USING DESCRIPTOR** *sqlda-name*
| **INTO** *variable-list* }
... [**PURGE**] [**BLOCK** *n*] [**ARRAY** *fetch-count*]
... **INTO** *variable-list*
... **IQ CACHE** *row-count*

Parameters

cursor-name:
identifier or host variable

sqlda-name:
identifier

host-variable-list:
may contain indicator variables

row-count:
number or host variable

fetch-count:
integer or host variable

Examples

- An Embedded SQL example:

```
EXEC SQL DECLARE cur_employee CURSOR FOR
SELECT emp_id, emp_lname FROM employee ;
EXEC SQL OPEN cur_employee;
EXEC SQL FETCH cur_employee
INTO :emp_number, :emp_name:indicator;
```

- A procedure example:

```
BEGIN
    DECLARE cur_employee CURSOR FOR
       SELECT emp_lname
       FROM employee ;
```

```
    DECLARE name CHAR(40) ;
    OPEN cur_employee;
    LOOP
       FETCH NEXT cur_employee into name ;
        ...
    END LOOP
    CLOSE cur_employee;
END
```

Usage

FETCH retrieves one row from the named cursor.

The ARRAY clause allows *wide fetches*, which retrieve more than one row at a time, and which might improve performance.

The cursor must have been previously opened.

One row from the result of SELECT is put into the variables in the variable list. The correspondence from the select list to the host variable list is one-to-one.

One or more rows from the result of SELECT are put either into the variables in the variable list or into the program data areas described by the named SQLDA. In either case, the correspondence from the select list to either the host variable list or the SQLDA descriptor array is one-to-one.

The INTO clause is optional. If it is not specified, then FETCH positions the cursor only (see the following paragraphs).

An optional positional parameter can be specified that allows the cursor to be moved before a row is fetched. The default is NEXT, which causes the cursor to be advanced one row before the row is fetched. PRIOR causes the cursor to be backed up one row before fetching.

RELATIVE positioning is used to move the cursor by a specified number of rows in either direction before fetching. A positive number indicates moving forward and a negative number indicates moving backwards. Thus, a NEXT is equivalent to RELATIVE 1 and PRIOR is equivalent to RELATIVE -1. RELATIVE 0 retrieves the same row as the last fetch statement on this cursor.

The ABSOLUTE positioning parameter is used to go to a particular row. A zero indicates the position before the first row . See Chapter 8, “Using Procedures and Batches” in the *Sybase IQ System Administration Guide*.

A one (1) indicates the first row, and so on. Negative numbers are used to specify an absolute position from the end of the cursor. A negative one (-1) indicates the last row of the cursor. FIRST is a short form for ABSOLUTE 1. LAST is a short form for ABSOLUTE -1.

Note Sybase IQ does not handle the FIRST, LAST, ABSOLUTE, and negative RELATIVE options very efficiently, so there is a performance impact when using them.

OPEN initially positions the cursor before the first row.

If the fetch includes a positioning parameter and the position is outside the allowable cursor positions, then the SQLE_NOTFOUND warning is issued.

The IQ CACHE clause specifies the maximum number of rows buffered in the FIFO queue. If you do not specify a value for it, the value of the CURSOR_WINDOW_ROWS database option is used. The default setting of CURSOR_WINDOW_ROWS is 200.

Using the FETCH statement in Embedded SQL

The following clauses are for use in Embedded SQL only:

- USING DESCRIPTOR *sqlda-name*
- INTO *host-variable-list*
- PURGE
- BLOCK n
- ARRAY fetch-count
- Use of *host-variable* in cursor-name and row-count.

DECLARE CURSOR must appear before FETCH in the C source code, and the OPEN statement must be executed before FETCH. If a host variable is being used for the cursor name, then the DECLARE statement actually generates code and thus must be executed before FETCH.

In the multiuser environment, rows can be fetched by the client more than one at a time. This is referred to as block fetching or multirow fetching. The first fetch causes several rows to be sent back from the server. The client buffers these rows and subsequent fetches are retrieved from these buffers without a new request to the server.

The BLOCK clause gives the client and server a hint as to how many rows may be fetched by the application. The special value of 0 means the request is sent to the server and a single row is returned (no row blocking).

The PURGE clause causes the client to flush its buffers of all rows and then send the fetch request to the server. This fetch request may return a block of rows.

If the SQLSTATE_NOTFOUND warning is returned on the fetch, then the *sqlerrd[2]* field of the SQLCA (SQLCOUNT) contains the number of rows that the attempted fetch exceeded the allowable cursor positions. (A cursor can be on a row, before the first row or after the last row.) The value is 0 if the row was not found but the position is valid, for example, executing FETCH RELATIVE 1 when positioned on the last row of a cursor. The value is positive if the attempted fetch was further beyond the end of the cursor, and negative if the attempted fetch was further before the beginning of the cursor.

After successful execution of the FETCH statement, the *sqlerrd[1]* field of the SQLCA (SQLIOCOUNT) is incremented by the number of input/output operations required to perform the fetch. This field is actually incremented on every database statement.

To use wide fetches in Embedded SQL, include the FETCH statement in your code as follows:

```
EXEC SQL FETCH . . . ARRAY nnn
```

where *ARRAY nnn* is the last item of the FETCH statement. The fetch count *nnn* can be a host variable. The SQLDA must contain nnn * (columns per row) variables. The first row is placed in SQLDA variables 0 to (columns per row)-1, and so on.

The server returns in SQLCOUNT the number of records fetched and always returns a SQLCOUNT greater than zero unless there is an error. Older versions of the server only return a single row and the SQLCOUNT is set to zero. Thus a SQLCOUNT of zero with no error condition indicates one valid row has been fetched.

Side effects

None.

Standards

- **SQL92** Entry-level feature. Use in procedures is a Persistent Stored Module feature.
- **Sybase** Supported in Adaptive Server Enterprise.

Permissions

The cursor must be opened and the user must have SELECT permission on the tables referenced in the declaration of the cursor.

See also

CURSOR_WINDOW_ROWS option on page 62

DECLARE CURSOR statement [ESQL] [SP] on page 516

OPEN statement [ESQL] [SP] on page 603

PREPARE statement [ESQL] on page 611

FOR statement

Description

Repeats the execution of a statement list once for each row in a cursor.

Syntax

[*statement-label*:]
... **FOR** *for-loop-name* **AS** *cursor-name*
... **CURSOR FOR** *statement*
... [{ **FOR UPDATE** | **FOR READ ONLY** }]
... **DO** *statement-list*
... **END FOR** [*statement-label*]

Examples

The following fragment illustrates the use of the FOR loop:

```
FOR names AS curs CURSOR FOR
SELECT emp_lname
FROM employee
DO
    CALL search_for_name( emp_lname );
END FOR;
```

Usage

FOR is a control statement that lets you execute a list of SQL statements once for each row in a cursor. The FOR statement is equivalent to a compound statement with a DECLARE for the cursor and a DECLARE of a variable for each column in the result set of the cursor followed by a loop that fetches one row from the cursor into the local variables and executes *statement-list* once for each row in the cursor.

The name and data type of the local variables that are declared are derived from the *statement* used in the cursor. With a SELECT statement, the data type is the data type of the expressions in the select list. The names are the select list item aliases where they exist; otherwise, they are the names of the columns. Any select list item that is not a simple column reference must have an alias. With a CALL statement, the names and data types are taken from the RESULT clause in the procedure definition.

The LEAVE statement can be used to resume execution at the first statement after the END FOR. If the ending *statement-label* is specified, it must match the beginning *statement-label*.

Side effects

None.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** Not supported by Adaptive Server Enterprise.

Permissions

None

See also

DECLARE CURSOR statement [ESQL] [SP] on page 516

FETCH statement [ESQL] [SP] on page 547

LEAVE statement on page 578

LOOP statement on page 598

FORWARD TO statement

Description

Sends native syntax to a remote server.

Syntax

Syntax 1

FORWARD TO *server-name* { *sql-statement* }

Syntax 2

FORWARD TO [*server-name*]

Examples

Shows a passthrough session with the remote server ase_prod:

```
FORWARD TO aseprod
SELECT * from titles
SELECT * from authors
FORWARD TO
```

Usage

The FORWARD TO statement enables users to specify the server to which a passthrough connection is required. The statement can be used in two ways:

- To send a statement to a remote server (Syntax 1)
- To place Sybase IQ into passthrough mode for sending a series of statements to a remote server (Syntax 2)

When establishing a connection to *server-name* on behalf of the user, the server uses:

- A remote login alias set using CREATE EXTERNLOGIN

- If a remote login alias is not set up, the name and password used to communicate with Sybase IQ.

If the connection cannot be made to the server specified, the reason is contained in a message returned to the user.

After statements are passed to the requested server, any results are converted into a form that can be recognized by the client program.

server-name is the name of the remote server.

sql-statement is a command in the remote server's native syntax. The command or group of commands is enclosed in curly braces ({}).

When you specify a *server_name*, but do not specify a statement in the FORWARD TO query, your session enters passthrough mode, and all subsequent queries are passed directly to the remote server. To turn passthrough mode off, issue FORWARD TO without a *server_name* specification.

Side effects

The remote connection is set to AUTOCOMMIT (unchained) mode for the duration of the FORWARD TO session. Any work that was pending prior to the FORWARD TO statement is automatically committed.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions

None.

See also

CREATE SERVER statement on page 494

FROM clause

Description

Specifies the database tables or views involved in a SELECT statement.

Syntax

... **FROM** *table-expression* [, ...]

Parameters

table-expression:
{ *table-spec*
| *table-expression join-type table-spec* [ON *condition*]
| (*table-expression* [, ...]) }

table-spec:
{ [*userid.*] *table-name* [[AS] *correlation-name*]
| *select-statement* [AS *correlation-name* (*column-name* [, ...])] }

join-type:
{ CROSS JOIN
| [NATURAL | KEY] JOIN
| [NATURAL | KEY] INNER JOIN
| [NATURAL | KEY] LEFT OUTER JOIN
| [NATURAL | KEY] RIGHT OUTER JOIN
| [NATURAL | KEY] FULL OUTER JOIN }

Examples

- The following are valid FROM clauses:

```
...
FROM employee
...
...
FROM employee NATURAL JOIN department
...
...
FROM customer
KEY JOIN sales_order
KEY JOIN sales_order_items
KEY JOIN product
...
```

- The following query illustrates how to use derived tables in a query:

```
SELECT lname, fname, number_of_orders
FROM customer JOIN
     ( SELECT cust_id, count(*)
       FROM sales_order
        GROUP BY cust_id )
     AS sales_order_counts ( cust_id,
                             number_of_orders )
ON ( customer.id = sales_order_counts.cust_id )
WHERE number_of_orders > 3
```

Usage

The SELECT statement requires a table list to specify which tables are used by the statement.

Note Although this description refers to tables, it also applies to views unless otherwise noted.

The FROM table list creates a result set consisting of all the columns from all the tables specified. Initially, all combinations of rows in the component tables are in the result set, and the number of combinations is usually reduced by join conditions and/or WHERE conditions.

The *join-type* keywords are described in Table 6-8.

Table 6-8: FROM clause join-type keywords

***join-type* keyword**	**Description**
CROSS JOIN	Returns the Cartesian product (cross product) of the two source tables
NATURAL JOIN	Compares for equality all corresponding columns with the same names in two tables (a special case equijoin; columns are of same length and data type)
KEY JOIN	Restricts foreign-key values in the first table to be equal to the primary-key values in the second table
INNER JOIN	Discards all rows from the result table that do not have corresponding rows in both tables
LEFT OUTER JOIN	Preserves unmatched rows from the left table, but discards unmatched rows from the right table
RIGHT OUTER JOIN	Preserves unmatched rows from the right table, but discards unmatched rows from the left table
FULL OUTER JOIN	Retains unmatched rows from both the left and the right tables

Do not mix comma-style joins and keyword-style joins in the FROM clause. The same query can be written two ways, each using *one* of the join styles. The ANSI syntax keyword style join is preferable.

The following query uses a comma-style join:

```
SELECT *
  FROM product pr, sales_order so, sales_order_items si
  WHERE pr.prod_id = so.prod_id
    AND pr.prod_id = si.prod_id;
```

The same query can use the preferable keyword-style join:

```
SELECT *
  FROM product pr INNER JOIN sales_order so
    ON (pr.prod_id = so.prod_id)
  INNER JOIN sales_order_items si
    ON (pr.prod_id = si.prod_id);
```

The ON clause filters the data of inner, left, right, and full joins. Cross joins do not have an ON clause. In an inner join, the ON clause is equivalent to a WHERE clause. In outer joins, however, the ON and WHERE clauses are different. The ON clause in an outer join filters the rows of a cross product and then includes in the result the unmatched rows extended with nulls. The WHERE clause then eliminates rows from both the matched and unmatched rows produced by the outer join. You must take care to ensure that unmatched rows you want are not eliminated by the predicates in the WHERE clause.

You cannot use subqueries inside an outer join ON clause.

For information on writing Transact-SQL compatible joins, see Appendix A, "Compatibility with Other Sybase Databases".

Tables owned by a different user can be qualified by specifying the *userid*. Tables owned by groups to which the current user belongs are found by default without specifying the user ID.

The correlation name is used to give a temporary name to the table for this SQL statement only. This is useful when referencing columns that must be qualified by a table name but the table name is long and cumbersome to type. The correlation name is also necessary to distinguish between table instances when referencing the same table more than once in the same query. If no correlation name is specified, then the table name is used as the correlation name for the current statement.

If the same correlation name is used twice for the same table in a table expression, that table is treated as if it were only listed once. For example, in:

```
SELECT *
FROM sales_order
KEY JOIN sales_order_items,
sales_order
KEY JOIN employee
```

The two instances of the sales_order table are treated as one instance that is equivalent to:

```
SELECT *
FROM sales_order_items
KEY JOIN sales_order
KEY JOIN employee
```

By contrast, the following is treated as two instances of the Person table, with different correlation names HUSBAND and WIFE.

```
SELECT *
FROM Person HUSBAND, Person WIFE
```

You can supply a SELECT statement instead of one or more tables or views in the FROM clause, letting you use groups on groups, or joins with groups, without creating a view. This use of SELECT statements is called derived tables.

Join columns require like data types for optimal performance.

Depending on the query, Sybase IQ allows between 16 and 64 tables in the FROM clause with the optimizer turned on; however, performance might suffer if you have more than 16 to 18 tables in the FROM clause in very complex queries.

Note If you omit the FROM clause, or if all tables in the query are in the SYSTEM dbspace, the query is processed by Adaptive Server Anywhere instead of Sybase IQ and might behave differently, especially with respect to syntactic and semantic restrictions and the effects of option settings. See the Adaptive Server Anywhere documentation for rules that might apply to processing.

If you have a query that does not require a FROM clause, you can force the query to be processed by Sybase IQ by adding the clause "FROM iq_dummy," where iq_dummy is a one-row, one-column table that you create in your database.

Side effects

None.

Standards

- **SQL92** Entry-level feature.
- **Sybase** The JOIN clause is not supported in some versions of Adaptive Server Enterprise. Instead, you must use the WHERE clause to build joins.

Permissions

Must be connected to the database.

See also

DELETE statement on page 525

SELECT statement on page 632

"Search conditions" on page 189

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

GET DESCRIPTOR statement [ESQL]

Description

Retrieves information about variables within a descriptor area, or retrieves actual data from a variable in a descriptor area.

Syntax

GET DESCRIPTOR *descriptor-name*
... { *hostvar* = **COUNT** } | **VALUE** *n assignment* [, ...] }

Parameters

assignment:
hostvar = { TYPE | LENGTH | PRECISION | SCALE | DATA
| INDICATOR | NAME | NULLABLE | RETURNED_LENGTH }

Examples

For an example, see ALLOCATE DESCRIPTOR statement [ESQL] on page 394.

Usage

The value *n* specifies the variable in the descriptor area about which information is retrieved. Type checking is performed when doing GET ... DATA to ensure that the host variable and the descriptor variable have the same data type.

If an error occurs, it is returned in the SQLCA.

Side effects

None.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions

None

See also

DEALLOCATE DESCRIPTOR statement [ESQL] on page 514

SET DESCRIPTOR statement [ESQL] on page 646

GOTO statement [T-SQL]

Description

Branches to a labeled statement.

Syntax

label:
GOTO *label*

Examples

The following Transact-SQL batch prints the message “yes” on the server window four times:

```
declare @count smallint
select @count = 1
restart:
```

```
print 'yes'
select @count = @count + 1
while @count <=4
 goto restart
```

Usage

Any statement in a Transact-SQL procedure or batch can be labeled. The label name is a valid identifier followed by a colon. In the GOTO statement, the colon is not used.

Side effects

None.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** Adaptive Server Enterprise supports the GOTO statement.

Permissions

None.

GRANT statement

Description

Gives permissions to specific users and creates new user IDs.

Syntax

Syntax 1

GRANT CONNECT TO *userid* [, ...] **IDENTIFIED BY** *password* [, ...]

Syntax 2

GRANT
{ **DBA**
| **GROUP**
| **MEMBERSHIP IN GROUP** *userid* [, ...]
| **RESOURCE** | **ALL** }
... **TO** *userid* [, ...]

Syntax 3

GRANT
{ **ALL** [**PRIVILEGES**]
| **ALTER**
| **DELETE**
| **INSERT**
| **REFERENCES** [(*column-name* [, ...])]
| **SELECT** [(*column-name* [, ...])]
| **UPDATE** [(*column-name*,...)]
... **ON** [*owner*.]*table-name* **TO** *userid* [, ...] [**WITH GRANT OPTION**]

Syntax 4

GRANT EXECUTE ON [*owner*.]*procedure-name* **TO** *userid* [, ...]

Syntax 5

GRANT INTEGRATED LOGIN TO *user_profile_name* [, ...] **AS USER** *userid*

Examples

- Makes two new users for the database:

```
GRANT
CONNECT TO Laurel, Hardy
IDENTIFIED BY Stan, Ollie
```

- Grants permissions on the employee table to user Laurel:

```
GRANT
SELECT, INSERT, DELETE
ON employee
TO Laurel
```

- Allows the user Hardy to execute the Calculate_Report procedure:

```
GRANT
EXECUTE ON Calculate_Report
TO Hardy
```

Usage

The GRANT statement is used to grant database permissions to individual user IDs and groups. It is also used to create and delete users and groups.

Syntax 1 and 2 of the GRANT statement are used for granting special privileges to users as follows:

CONNECT TO userid,... Creates a new user. GRANT CONNECT can also be used by any user to change their own password.

To create a user with the empty string as the password, enter:

```
GRANT CONNECT TO userid IDENTIFIED BY ""
```

If you have DBA authority, you can change the password of any existing user with the following command:

```
GRANT CONNECT TO userid IDENTIFIED BY password
```

You can also use the same command to add a new user. For this reason, if you inadvertently enter the user ID of an existing user when you mean to add a new user, you are actually changing the password of the existing user. You do not receive a warning because this behavior is considered normal. This behavior differs from pre-version 12 Sybase IQ.

To avoid this situation, use the system procedures sp_addlogin and sp_adduser to add users. These procedures give you an error if you try to add an existing user ID, as in Adaptive Server Enterprise, and pre-version 12 Sybase IQ.

Note If Login Management is enabled for the database, you must use system procedures, not GRANT and REVOKE, to add and remove user IDs.

To create a user with no password, enter:

```
GRANT CONNECT TO userid
```

The user ID is not case sensitive.

By default, you can add users with GRANT CONNECT only on a multiplex write server. To enable GRANT CONNECT on query servers, you must set the database option MPX_LOCAL_SPEC_PRIV to change the default. For details, see "MPX_LOCAL_SPEC_PRIV option" on page 123.

A user with no password cannot connect to the database. This is useful when you are creating groups and you do not want anyone to connect to the group user ID.

The password must be a valid identifier, as described in "Identifiers" on page 177. Passwords have a maximum length of 255 bytes. If the database option VERIFY_PASSWORD_FUNCTION is set to a value other than the empty string, the GRANT CONNECT TO *userid* IDENTIFIED BY *password* statement calls the function identified by the option value. The function returns NULL to indicate that the password conforms to rules. If the VERIFY_PASSWORD_FUNCTION option is set, you can specify only one *usrid* and *password* with the GRANT CONNECT statement. For details, see "VERIFY_PASSWORD_FUNCTION option" on page 170.

The following are invalid for database user IDs and passwords:

- Names that begin with white space or single or double quotes
- Names that end with white space
- Names that contain semicolons

DBA Database Administrator authority gives a user permission to do anything. This is usually reserved for the person in the organization who is looking after the database.

GROUP Allows users to have members. See Chapter 12, "Managing User IDs and Permissions" in the *Sybase IQ System Administration Guide* for a complete description.

MEMBERSHIP IN GROUP userid,... Allows users to inherit table permissions from a group and to reference tables created by the group without qualifying the table name.

Syntax 3 of the GRANT statement is used to grant permission on individual tables or views. You can list the table permissions together, or specify ALL to grant all six permissions at once. The permissions have the following meaning:

RESOURCE Allows the user to create tables and views. In syntax 2, ALL is a synonym for RESOURCE, which is compatible with Adaptive Server Enterprise.

ALL In syntax 3, this grants all of the permissions outlined below.

ALTER Users can alter this table with the ALTER TABLE statement. This permission is not allowed for views.

DELETE Users can delete rows from this table or view.

INSERT Users can insert rows into the named table or view.

REFERENCES [(column-name,...)] Users can create indexes on the named tables, and foreign keys that reference the named tables. If column names are specified, then users can reference only those columns. REFERENCES permissions on columns cannot be granted for views, only for tables.

SELECT [(column-name,...)] Users can look at information in this view or table. If column names are specified, then the users can look at only those columns. SELECT permissions on columns cannot be granted for views, only for tables.

UPDATE [(column-name,...)] Users can update rows in this view or table. If column names are specified, users can update only those columns. UPDATE permissions on columns cannot be granted for views, only for tables. To update a table, users must have both SELECT and UPDATE permission on the table.

For example, to grant SELECT and UPDATE permissions on the employee table to user Laurel, enter:

```
GRANT
SELECT, UPDATE ( street )
ON employee
TO Laurel
```

If WITH GRANT OPTION is specified, then the named user ID is also given permission to GRANT the same permissions to other user IDs.

Syntax 4 of the GRANT statement is used to grant permission to execute a procedure.

Syntax 5 of the GRANT statement creates an explicit integrated login mapping between one or more Windows user profiles and an existing database user ID, allowing users who successfully log in to their local machine to connect to a database without having to provide a user ID or password.

Side effects

Automatic commit.

Standards

- **SQL92** Syntax 3 is an entry-level feature. Syntax 4 is a Persistent Stored Module feature. Other syntaxes are vendor extensions.
- **Sybase** Syntax 2 and 3 are supported in Adaptive Server Enterprise. The security model is different in Adaptive Server Enterprise and Sybase IQ, so other syntaxes differ.

Permissions

- For Syntax 1 or 2, one of the three following conditions must be met:
  - You are changing your own password using GRANT CONNECT
  - You are adding members to your own user ID
  - You have DBA authority.

  If you are changing another user's password, the other user cannot be connected to the database.
- For Syntax 3, one of the following conditions must be met:
  - You created the table
  - You have been granted permissions on the table with GRANT OPTION
  - You have DBA authority
- For Syntax 4, one of the following conditions must be met:
  - You created the procedure
  - You have DBA authority
- For Syntax 5, you must have DBA authority.

See also

REVOKE statement on page 628

HELP statement [DBISQL]

Description — Accesses help in the DBISQL environment.

Syntax — **HELP** [*topic*]

Usage — The HELP statement is used to enter the DBISQL interactive help facility. The *topic* for help can be optionally specified. If *topic* is not specified, the help system is entered at the index.

Side effects

None.

Standards
- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions — None.

IF statement

Description — Provides conditional execution of SQL statements.

Syntax
IF *search-condition* **THEN** *statement-list*
... [**ELSE IF** *search-condition* **THEN** *statement-list*]...
... [**ELSE** *statement-list*]
... **END IF**

Examples — The following procedure illustrates the use of the IF statement:

```
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35),
OUT TopValue INT)
BEGIN
    DECLARE err_notfound EXCEPTION
    FOR SQLSTATE '02000' ;
    DECLARE curThisCust CURSOR FOR
    SELECT company_name, CAST(
sum(sales_order_items.quantity *
    product.unit_price) AS INTEGER) VALUE
    FROM customer
    LEFT OUTER JOIN sales_order
    LEFT OUTER JOIN sales_order_items
    LEFT OUTER JOIN product
    GROUP BY company_name ;


    DECLARE ThisValue INT ;
```

```
    DECLARE ThisCompany CHAR(35) ;
    SET TopValue = 0 ;
    OPEN curThisCust ;
    CustomerLoop:
    LOOP
       FETCH NEXT curThisCust
       INTO ThisCompany, ThisValue ;
       IF SQLSTATE = err_notfound THEN
          LEAVE CustomerLoop ;
       END IF ;
       IF ThisValue > TopValue THEN
          SET TopValue = ThisValue ;
          SET TopCompany = ThisCompany ;
       END IF ;
    END LOOP CustomerLoop ;
    CLOSE curThisCust ;
END
```

Usage

The IF statement lets you conditionally execute the first list of SQL statements whose *search-condition* evaluates to TRUE. If no *search-condition* evaluates to TRUE, and an ELSE clause exists, the *statement-list* in the ELSE clause is executed. If no *search-condition* evaluates to TRUE, and there is no ELSE clause, the expression returns a NULL value.

Execution resumes at the first statement after the END IF.

When comparing variables to the single value returned by a SELECT statement inside an IF statement, you must first assign the result of the SELECT to another variable.

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

For information on the IF expression, see “Expressions” on page 179.

Side effects

None.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** The Transact-SQL IF statement has a slightly different syntax.

Permissions

None.

See also

BEGIN... END statement on page 422

IF statement [T-SQL]

Description

Provides conditional execution of a Transact-SQL statement, as an alternative to the Sybase IQ IF statement.

Syntax

IF *expression*
... *statement*
... [**ELSE** [**IF** *expression*] *statement*]...

Examples

- The following example illustrates the use of the Transact-SQL IF statement:

```
IF (SELECT max(id) FROM sysobjects) < 100
    RETURN
ELSE
    BEGIN
        PRINT "These are the user-created objects"
        SELECT name, type, id
        FROM sysobjects
        WHERE id < 100
END
```

- The following two statement blocks illustrate Transact-SQL and Sybase IQ compatibility:

```
/* Transact-SQL IF statement */
IF @v1 = 0
    PRINT '0'
ELSE IF @v1 = 1
    PRINT '1'
ELSE
    PRINT 'other'
/* IQ IF statement */
IF v1 = 0 THEN
    PRINT '0'
ELSEIF v1 = 1 THEN
    PRINT '1'
ELSE
    PRINT 'other'
END IF
```

Usage

The Transact-SQL IF conditional and the ELSE conditional each control the performance of only a single SQL statement or compound statement (between the keywords BEGIN and END).

In contrast to the Sybase IQ IF statement, the Transact-SQL IF statement has no THEN. The Transact-SQL version also has no ELSE IF or END IF keywords.

When comparing variables to the single value returned by a SELECT statement inside an IF statement, you must first assign the result of the SELECT to another variable.

Side effects

None.

Standards

- **SQL92** Transact-SQL extension.
- **Sybase** Adaptive Server Enterprise supports the Transact-SQL IF statement.

Permissions

None

INCLUDE statement [ESQL]

Description

Includes a file into a source program to be scanned by the SQL source language preprocessor.

Syntax

INCLUDE *filename*

Parameters

filename:
identifier

Usage

The INCLUDE statement is very much like the C preprocessor #include directive. However, the SQL preprocessor reads the given file, inserting its contents into the output C file. Thus, if an include file contains information that the SQL preprocessor requires, it should be included with the Embedded SQL INCLUDE statement.

Two file names are specially recognized: SQLCA and SQLDA. Any C program using Embedded SQL must contain the following statement before any Embedded SQL statements:

```
EXEC SQL INCLUDE SQLCA;
```

This statement must appear at a position in the C program where static variable declarations are allowed. Many Embedded SQL statements require variables (invisible to the programmer) which are declared by the SQL preprocessor at the position of the SQLCA include statement. The SQLDA file must be included if any SQLDAs are used.

Side effects

None.

Standards

- **SQL92** Entry-level feature.

- **Sybase** Supported by Open Client/Open Server.

Permissions None

INSERT statement

Description Inserts into a table either a single row (Syntax 1) or a selection of rows (Syntax 2) from elsewhere in the current database. Inserts a selection of rows from another database (Syntax 3) or lets users specify the setting for options at the remote server.

Syntax

Syntax 1

INSERT [**INTO**] [*owner*.]*table-name* [(*column-name* [, ...])]
... **VALUES** (*expression* ...)

Syntax 2

INSERT [**INTO**] [*owner*.]*table-name* [(*column-name* [, ...])]
... *insert-load-options*
... *select-statement*

Syntax 3

INSERT [**INTO**] [*owner*.]*table-name*[(*column-name* [, ...])]
... *insert-load-options*
[**LOCATION** '*servername.dbname*'
[*location-options*]]
... { *select-statement* }

Parameters

insert-load-options:
[LIMIT *number-of-rows*]
[NOTIFY *number-of-rows*]
[SKIP *number-of-rows*]
[START ROW ID *number*]

location-options:
[ENCRYPTED PASSWORD]
[PACKETSIZE *packet-size*]
[QUOTED_IDENTIFIER { ON | OFF }]

Examples

- Adds an Eastern Sales department to the database:

```
INSERT INTO department ( dept_id, dept_name )
VALUES ( 230, 'Eastern Sales' )
```

- Fills the table dept_head with the names of department heads and their departments:

```
INSERT INTO dept_head (name, dept)
   NOTIFY 20
   SELECT emp_fname || ' ' || emp_fname
   AS name,
   dept_name
FROM employee JOIN department
   ON emp_id = dept_head_id
```

- Inserts data from the l_shipdate and l_orderkey columns of the lineitem table from the Sybase IQ 11.5 database asiq11db.dba on the server detroit, into the corresponding columns of the lineitem table in the current database:

```
INSERT INTO lineitem
   (l_shipdate, l_orderkey)
   LOCATION 'detroit.asiqdb'
   PACKETSIZE 512
   ' SELECT l_shipdate, l_orderkey
FROM lineitem '
```

Usage

Syntax 1 allows the insertion of a single row with the specified expression values. If the list of column names is not specified, the values are inserted into the table columns in the order they were created (the same order as retrieved with SELECT *). The row is inserted into the table at an arbitrary position. (In relational databases, tables are not ordered.)

Syntax 2 allows the user to do mass insertion into a table with the results of a fully general SELECT statement. Insertions are done in an arbitrary order unless the SELECT statement contains an ORDER BY clause. The columns from the select list are matched ordinally with the columns specified in the column list, or sequentially in the order in which the columns were created.

Note The NUMBER(*) function is useful for generating primary keys with Syntax 2 of the INSERT statement (see Chapter 5, “SQL Functions”).

Syntax 3 INSERT...LOCATION is a variation of Syntax 2 that lets you insert data from an Adaptive Server Enterprise or Sybase IQ database. The *servername.dbname* identifies the server and database for the table in the FROM clause. The SELECT statement might be delimited by either curly braces or straight single quotation marks. (Curly braces represent the start and end of an escape sequence in the ODBC standard, and might generate errors in the context of ODBC.) To use Syntax 3, the Adaptive Server Enterprise server to which you are connecting must exist in the *interfaces* file on the local machine.

The following Open Client restrictions apply to queries using Syntax 3:

- You can insert a maximum of 2147483647 rows.
- You cannot use unsigned integer data.

Sybase IQ connects to the server and database you specify and returns the results from the queries in those tables to insert in the current database. If you omit the *server-name*, Sybase IQ ignores any *database-name* you might specify, since the only choice is the current database on the local server.

When Sybase IQ connects to the remote server, INSERT...LOCATION uses the remote login for the user ID of the current connection, if a remote login has been created with CREATE EXTERNLOGIN and the remote server has been defined with a CREATE SERVER statement. If the remote server is not defined or a remote login has not been created for the user ID of the current connection, Sybase IQ connects using the user ID and password of the current connection.

For example, user russid connects to the Sybase IQ database and executes the following statement:

```
INSERT local_SQL_Types LOCATION 'ase1.ase1db'
{SELECT int_col FROM SQL_Types};
```

On server ase1, there exists user ID ase1user with password sybase. The owner of the table SQL_Types is ase1user. The remote server is defined on the IQ server as follows:

```
CREATE SERVER ase1 CLASS 'ASEJDBC'
USING 'system1:4100';
```

The external login is defined on the IQ server as follows:

```
CREATE EXTERNLOGIN russid TO ase1 REMOTE LOGIN ase1user
IDENTIFIED BY sybase;
```

INSERT...LOCATION connects to the remote server ase1 using the user ID ase1user and the password sybase for user russid.

The ENCRYPTED PASSWORD parameter lets you specify the use of Open Client Library default password encryption when connecting to a remote server. If ENCRYPTED PASSWORD is specified and the remote server does not support Open Client Library default password encryption, an error is reported indicating that an invalid user ID or password was used. When used as a remote server, Sybase IQ does not support this password encryption.

The PACKETSIZE parameter specifies the TDS packet size in bytes. The default TDS packet size on most platforms is 512 bytes. If your application is receiving large amounts of text or bulk data across a network, then a larger packet size might significantly improve performance.

The value of *packet-size* must be a multiple of 512 either equal to the default network packet size or between the default network packet size and the maximum network packet size. The maximum network packet size and the default network packet size are multiples of 512 in the range 512 – 524288 bytes. The maximum network packet size is always greater than or equal to the default network packet size. See the *Adaptive Server Enterprise System Administration Guide, Volume 1* for more information on network packet size.

If INSERT...LOCATION PACKETSIZE *packet-size* is not specified or is specified as zero, then the default packet size value for the platform is used.

Note If you specify an incorrect packet size (for example 933, which is not a multiple of 512), the connection attempt fails with an Open Client ct_connect "Connection failed" error. Any unsuccessful connection attempt returns a generic "Connection failed" message. The Adaptive Server Enterprise error log might contain more specific information about the cause of the connection failure.

The QUOTED_IDENTIFIER parameter lets you specify the setting of the QUOTED_IDENTIFIER option on the remote server. The default setting is 'OFF'. You set QUOTED_IDENTIFIER to 'ON' only if any of the identifiers in the SELECT statement are enclosed in double quotes, as in the following example using 'c1':

```
INSERT INTO foo
LOCATION 'ase.database'
QUOTED_IDENTIFIER ON select "c1" from xxx;
```

While you are connected by INSERT...LOCATION, the IQ hostname and the program_name Sybase IQ appear in sysprocesses in the Adaptive Server Enterprise master database.

Sybase IQ does not support the Adaptive Server Enterprise data type TEXT, but you can execute INSERT...LOCATION (Syntax 3) from both an IQ CHAR or VARCHAR column whose length is greater than 255 bytes, and from an ASE database column of data type TEXT. ASE TEXT and IMAGE columns can be inserted into columns of other Sybase IQ data types, if Sybase IQ supports the internal conversion. All data inserted is silently right truncated at 32767 bytes.

Note If you use INSERT...LOCATION to insert data selected from a VARBINARY column, set the LOAD_MEMORY_MB option on the *local* database to limit memory used by the insert, and set ASE_BINARY_DISPLAY to OFF on the *remote* database.

INSERT...LOCATION (Syntax 3) does not support the use of variables in the SELECT statement.

Inserts can be done into views, provided the SELECT statement defining the view has only one table in the FROM clause and does not contain a GROUP BY clause, an aggregate function, or involve a UNION operation.

Character strings inserted into tables are always stored in the case they are entered, regardless of whether the database is case sensitive or not. Thus, a string “Value” inserted into a table is always held in the database with an uppercase V and the remainder of the letters lowercase. SELECT statements return the string as *Value*. If the database is not case-sensitive, however, all comparisons make *Value* the same as *value*, *VALUE*, and so on. Further, if a single-column primary key already contains an entry Value, an INSERT of value is rejected, as it would make the primary key not unique.

Whenever you execute an INSERT ... LOCATION statement, Sybase IQ loads the localization information needed to determine language, collation sequence, character set, and date/time format. If your database uses a nondefault locale for your platform, you must set an environment variable on your local client to ensure that Sybase IQ loads the correct information.

If you set the LC_ALL environment variable, Sybase IQ uses its value as the locale name. If LC_ALL is not set, Sybase IQ uses the value of the LANG environment variable. If neither variable is set, Sybase IQ uses the default entry in the locales file. For an example, see “Setting locales” in Chapter 11, “International Languages and Character Sets” in the *Sybase IQ System Administration Guide*.

The LIMIT option specifies the maximum number of rows to insert into the table from a query. The default is 0 for no limit. The maximum is 2GB -1.

The NOTIFY option specifies that you be notified with a message each time the number of rows are successfully inserted into the table. The default is every 100,000 rows.

The SKIP option lets you define a number of rows to skip at the beginning of the input tables for this insert. The default is 0.

The START ROW ID option specifies the record identification number of a row in the IQ table where it should start inserting. This option is used for *partial-width* inserts, which are inserts into a subset of the columns in the table. By default, new rows are inserted wherever there is space in the table, and each insert starts a new row. Partial-width inserts need to start at an existing row. They also need to insert data from the source table into the destination table positionally by column, so you must specify the destination columns in the same order as their corresponding source columns. The default is 0. For more information about partial-width inserts, see Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

Note Use the START ROW ID option for partial-width inserts only. If the columns being loaded already contain data, the insert fails.

An INSERT on a multicolumn index must include all columns of the index.

Sybase IQ supports column DEFAULT values for INSERT...VALUES, INSERT...SELECT, and INSERT...LOCATION. If a DEFAULT value is specified for a column, this DEFAULT value is used as the value of the column in any INSERT (or LOAD) statement that does not specify a value for the column.

For more information on the use of column DEFAULT values with inserts, see "Using column defaults" in Chapter 9, "Ensuring Data Integrity" in the *Sybase IQ System Administration Guide*.

Side effects

None.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Adaptive Server Enterprise (excluding the *insert-load-options*).

Permissions

Must have INSERT permission on the table.

See also

DELETE statement on page 525

LOAD TABLE statement on page 580

SYNCHRONIZE JOIN INDEX statement on page 657

"Using the INSERT statement" in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*

INSTALL statement

Description

Makes Java classes available for use within a database.

Syntax

INSTALL JAVA [*install-mode*] [**JAR** *jar-name*] **FROM** *source*

Parameters

install-mode:
{ NEW | UPDATE }

source:
{ FILE *filename* | URL *url-value* }

Examples

- Installs the user-created Java class named "Demo" by providing the file name and location of the class:

  ```
  INSTALL JAVA NEW
  FROM FILE 'D:\JavaClass\Demo.class'
  ```

  After installation, the class is referenced using its name. Its original file path location is no longer used. For example, the following statement uses the class installed in the previous statement:

  ```
  CREATE VARIABLE d Demo
  ```

  If the Demo class was a member of the package sybase.work, the fully qualified name of the class must be used, for example:

  ```
  CREATE VARIABLE d sybase.work.Demo
  ```

- Installs all the classes contained in a zip file, and associates them within the database with a JAR file name:

  ```
  INSTALL JAVA
  JAR 'Widgets'
  FROM FILE 'C:\Jars\Widget.zip'
  ```

  Again, the location of the zip file is not retained, and classes must be referenced using the fully qualified class name (package name and class name). The zip file must be an uncompressed JAR file.

Usage

Install mode Specifying an install mode of NEW requires that the referenced Java classes be new classes, rather than updates of currently installed classes. An error occurs if a class with the same name exists in the database and the NEW install mode is used.

Specifying UPDATE specifies that the referenced Java classes may include replacements for Java classes already installed in the given database.

Connection must be dropped for update to take effect Updating a Java class installed in a database takes effect immediately. However, the connection used to execute the INSTALL JAVA UPDATE statement has access only to the older version of the Java class until the connection is dropped.

Note A client application executing this statement should drop the database connection used to execute the statement and reconnect to get access to the latest version.

This applies to the DBISQL utility also. If you update a Java class by executing the INSTALL statement from DBISQL, the new version is not available until you disconnect from the database engine or server and reconnect.

If install mode is omitted, the default is NEW.

JAR If this is specified, the *file-name* or *text-pointer* must designate a JAR file or a column containing a JAR. JAR files typically have extensions of *.jar* or *.zip*.

Installed JAR and zip files can be compressed or uncompressed. However, JAR files produced by the Sun JDK *jar* utility are not supported. Files produced by other zip utilities are supported.

If the JAR option is specified, then the JARis retained as a JAR after the classes that it contains have been installed. That JAR is the associated JARof each of those classes. The set of JARs installed in a database with the JAR option are called the retained JARs of the database.

Retained JARs are referenced in INSTALL and REMOVE statements. Retained JARs have no effect on other uses of Java-SQL classes. Retained JARs are used by the SQL system for requests by other systems for the class associated with given data. If a requested class has an associated JAR, the SQL system can supply that JAR, rather than the individual class.

jar-name is a character string value of length up to 255 bytes. *jar-name* is used to identify the retained JAR in subsequent INSTALL, UPDATE, and REMOVE statements.

source Specifies the location of the Java classes to be installed.

The formats supported for *file-name* include fully qualified file names, such as '*c:\libs\jarname.jar*' and '*/usr/u/libs/jarname.jar*', and relative file names, which are relative to the current working directory of the database server.

The *filename* must identify either a class file, or a JAR file.

Class availability

The class definition for each class is loaded by each connection's VM the first time that class is used. When you INSTALL a class, the VM on your connection is implicitly restarted. Therefore, you have immediate access to the new class, whether the INSTALL has an *install-mode* of NEW or UPDATE.

For other connections, the new class is loaded the next time a VM accesses the class for the first time. If the class is already loaded by a VM, that connection does not see the new class until the VM is restarted for that connection (for example, with a STOP JAVA and START JAVA).

Standards

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

Permissions

- Requires DBA permissions to execute the INSTALL statement.
- All installed classes can be referenced in any way by any user.

See also REMOVE statement on page 619

IQ UTILITIES statement

Description

Collects statistics on the buffer caches for a Sybase IQ database.

Syntax

IQ UTILITIES { **MAIN** | **PRIVATE** }
[**INTO**] *table-name*
{ **START MONITOR** ['*monitor-options*']
| **STOP MONITOR** }

Parameters

monitor-options:
{ -summary |
{ -append | -truncate }
-bufalloc |
-cache |
-cache_by_type |
-contention |
-debug |
-file_suffix *suffix*|
-io |
-interval *seconds* |
-threads }...

Examples

Starts the buffer cache monitor and records activity for the IQ temp buffer cache:

```
IQ UTILITIES PRIVATE INTO monitor START MONITOR '-cache
-interval 20'
```

Usage

START MONITOR starts the IQ buffer cache monitor. For START and STOP MONITOR, the *table_name* is a dummy table. You can specify any IQ base or temporary table, although it is best to have a table that you use only for monitoring. Results go to a text file, *dbname.connection#-main-iqmon* for MAIN buffer cache results, or *dbname.connection#-temp-iqmon* for PRIVATE (Temp) buffer cache results. Running the monitor again from the same database and connection number overwrites previous results. To set the directory location of the monitor output file, set the MONITOR_OUTPUT_DIRECTORY option.

The *monitor-options* define the content and frequency of results. You can specify more than one, and they must be enclosed with quotation marks.

- -summary displays summary information for both the main and temp (private) buffer caches. This option is the default.
- -append | -truncate appends to the existing output file or truncates the existing output file, respectively. Truncate is the default.
- -bufalloc displays information on the main or temp buffer allocator, which reserves space in the buffer cache for objects like sorts, hashes, and bitmaps.
- -cache displays main or temp buffer cache activity in detail.
- -cache_by_type produces the same results as -cache, but broken down by IQ page type. This format is used mainly to supply information to Sybase Technical Support.
- -contention displays many key buffer cache and memory manager locks.
- -debug displays all the information that is available to the performance monitor, whether or not there is a standard display mode that covers the same information. This option is used mainly to supply information to Sybase Technical Support.
- -file_suffix *suffix* creates a monitor output file named `<dbname>.<connid>-<main_or_temp>-<suffix>`. If you do not specify a suffix, it defaults to *iqmon*.
- -io displays main or temp buffer cache I/O rates and data compression ratios.

- -interval specifies the reporting interval in seconds. The default is every 60 seconds. The minimum is every 2 seconds.
- -threads displays information about processing threads.

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not supported in Adaptive Server Enterprise.

Permissions

None

See also

MONITOR_OUTPUT_DIRECTORY option on page 121

Chapter 6, "Monitoring and Tuning Performance" in the *Sybase IQ Performance and Tuning Guide*, for examples of monitor results

Chapter 8, "Using Procedures and Batches" in *Sybase IQ System Administration Guide* for advanced use of IQ UTILITIES to create procedures that extend the functionality of Sybase IQ system stored procedures

LEAVE statement

Description

Continues execution by leaving a compound statement or LOOP.

Syntax

LEAVE *statement-label*

Examples

- The following fragment shows how the LEAVE statement is used to leave a loop:

```
SET i = 1;
lbl:
LOOP
    INSERT
    INTO Counters ( number )
    VALUES ( i ) ;
    IF i >= 10 THEN
       LEAVE lbl ;
    END IF ;
    SET i = i + 1
END LOOP lbl
```

- The following fragment uses LEAVE in a nested loop:

```
outer_loop:
```

```
LOOP
    SET i = 1;
    inner_loop:
    LOOP
        ...
        SET i = i + 1;
        IF i >= 10 THEN
            LEAVE outer_loop
        END IF
    END LOOP inner_loop
END LOOP outer_loop
```

Usage

LEAVE is a control statement that lets you leave a labeled compound statement or a labeled loop. Execution resumes at the first statement after the compound statement or loop.

The compound statement that is the body of a procedure has an implicit label that is the same as the name of the procedure.

Side effects

None.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** Not supported in Adaptive Server Enterprise. The break statement provides a similar feature for Transact-SQL compatible procedures.

Permissions

None

See also

BEGIN... END statement on page 422

FOR statement on page 551

LOOP statement on page 598

LOAD TABLE statement

Description

Imports data into a database table from an external ASCII-format file.

Syntax

LOAD [**INTO**] **TABLE** [*owner*].*table-name*
... (*load-specification* [, ...])
... **FROM** { '*filename-string*' | *filename-variable* } [, ...]
... [**CHECK CONSTRAINTS** { **ON** | **OFF** }]
... [**DEFAULTS** { **ON** | **OFF** }]
... **QUOTES OFF**
... **ESCAPES OFF**
... [**FORMAT** { **'ascii'** | **'binary'** }]
... [**DELIMITED BY** '*string*']
... [**STRIP** { **ON** | **OFF** }]
... [**WITH CHECKPOINT** { **ON** | **OFF** }]
... [{ **BLOCK FACTOR** *number* | **BLOCK SIZE** *number* }]
... [**BYTE ORDER** { **NATIVE** | **HIGH** | **LOW** }]
... [**LIMIT** *number-of-rows*]
... [**NOTIFY** *number-of-rows*]
... [**ON FILE ERROR** { **ROLLBACK** | **FINISH** | **CONTINUE**}]
... [**PREVIEW** { **ON** | **OFF** }]
... [**ROW DELIMITED BY** '*delimiter-string*']
... [**SKIP** *number-of-rows*]
... [**WORD SKIP** *number*]
... [**START ROW ID** *number*]
... [**UNLOAD FORMAT**]
... [**IGNORE CONSTRAINT** *constrainttype* [, ...]]
... [**MESSAGE LOG** '*string*' **ROW LOG** '*string*' [**ONLY LOG** *logwhat* [, ...]]
... [**LOG DELIMITED BY** '*string*']

Parameters

load-specification:
{ *column-name* [*column-spec*]
| FILLER (*filler-type*) }

column-spec:
{ ASCII (*input-width*)
| BINARY [WITH NULL BYTE]
| PREFIX { 1 | 2 | 4 }
| '*delimiter-string*'
| DATE (*input-date-format*)
| DATETIME (*input-datetime-format*) }
[NULL ({ BLANKS | ZEROS | '*literal*', ... })]

filler-type:
{ *input-width*
| PREFIX { 1 | 2 | 4 }
| '*delimiter-string*' }

constrainttype:
{ CHECK *integer* | UNIQUE *integer*
| NULL *integer*
| FOREIGN KEY *integer*
| DATA VALUE *integer*
| ALL *integer* }

logwhat:
{ CHECK | ALL | NULL | UNIQUE | DATA VALUE | FOREIGN KEY | WORD }

Examples

- Loads data from one file into the product table on a Windows system. A tab is used as the column delimiter following the description and color columns.

```
LOAD TABLE product
( id   ASCII(6),
FILLER(1),
name   ASCII(15),
FILLER(1),
description   '\x09',
size   ASCII(2),
FILLER(1),
color   '\x09',
quantity   PREFIX 2,
unit_price   PREFIX 2,
FILLER(2) )
FROM 'C:\\mydata\\source1.dmp'
QUOTES OFF
ESCAPES OFF
BYTE ORDER LOW
NOTIFY 1000
```

- Loads data from two files into the product_new table (which allows NULL values) on a UNIX system. The tab character is the default column delimiter, and the newline character is the row delimiter.

```
LOAD TABLE product_new
( id,
name,
description,
size,
color   '\x09'   NULL( 'null', 'none', 'na' ),
quantity   PREFIX 2,
unit_price   PREFIX 2 )
FROM '/s1/mydata/source2.dump', '/s1/mydata/
source3.dump'
QUOTES OFF
```

```
ESCAPES OFF
BLOCKSIZE 100000
FORMAT ascii
DELIMITED BY '\x09'
ON FILE ERROR CONTINUE
ROW DELIMITED BY '\n'
```

- Ignores 10 word-length violations; on the 11th, diplays the new error and rolls back the load:

```
load table PTAB1(
            ck1                ','  null ('NULL') ,
            ck3fk2c2           ','  null ('NULL') ,
            ck4                ','  null ('NULL') ,
            ck5                ','  null ('NULL') ,
            ck6c1              ','  null ('NULL') ,
            ck6c2              ','  null ('NULL') ,
            rid                ','  null ('NULL')  )
FROM 'ri_index_selfRI.inp'
            row delimited by '\n'
            LIMIT 14   SKIP 10
            IGNORE CONSTRAINT UNIQUE 2, FOREIGN KEY 8
            word skip 10 quotes off escapes off strip
            off
```

Usage

The LOAD TABLE statement allows efficient mass insertion into a database table from a file with ASCII or binary data.

The LOAD TABLE options also let you control load behavior when integrity constraints are violated and to log information about the violations.

If WITH CHECKPOINT ON is not specified, the file used for loading must be retained in case recovery is required. If WITH CHECKPOINT ON is specified, a checkpoint is carried out after loading, and recovery is guaranteed even if the data file is then removed from the system.

You can use LOAD TABLE on a temporary table, but the temporary table must have been declared with ON COMMIT PRESERVE ROWS, or the next COMMIT removes the rows you have loaded.

You can also specify more than one file to load data. In the FROM clause, you specify each *filename-string* separated by commas. However, Sybase IQ cannot guarantee that all the data can be loaded because of memory constraints. If memory allocation fails, the entire load transaction is rolled back. The files are read one at a time, and they are processed in a left-to-right order as specified in the FROM clause. Any SKIP or LIMIT value only applies in the beginning of the load, not for each file.

Note When loading a multiplex database, use absolute (fully qualified) paths in all file names. Do not use relative path names.

Sybase IQ supports loading from both ASCII and binary data, and it supports both fixed- and variable-length formats. To handle all of these formats, you must supply a *load-specification* to tell Sybase IQ what kind of data to expect from each "column" or field in the source file. The *column-spec* lets you define the following formats:

- ASCII with a fixed length of bytes. The *input-width* value is an integer value indicating the fixed width in bytes of the input field in every record.
- Binary fields that use a number of PREFIX bytes (1, 2, or 4) to specify the length of the binary input.

  If the data is unloaded using the extraction facility with the TEMP_EXTRACT_BINARY option set ON, you *must* use the BINARY WITH NULL BYTE parameter for each column when you load the binary data.
- Variable-length characters delimited by a separator. You specify the terminator as hexadecimal ASCII characters. The *delimiter-string* can be any string of up to 4 characters, including any combination of printable characters, and any 8-bit hexadecimal ASCII code that represents a nonprinting character. For example, specify:
  - '\x09' to represent a tab as the terminator.
  - '\x00' for a null terminator (no visible terminator as in "C" strings).

- '\x0a' for a newline character as the terminator. You can also use the special character combination of '\n' for newline.

Note The delimiter string can be from 1 to 4 characters long, but you can specify only a single character in the DELIMITED BY clause.

- DATE or DATETIME string as ASCII characters. You must define the *input-date-format* or *input-datetime-format* of the string using one of the corresponding formats for the date and datetime data types supported by Sybase IQ. Use DATE for date values and DATETIME for datetime and time values.

Table 6-9: Formatting dates and times

Option	Meaning
yyyy or YYYY yy or YY	Represents number of year. Default is current year.
mm or MM	Represents number of month. Always use leading zero or blank for number of the month where appropriate, for example, '05' for May. DATE value must include a month. For example, if the DATE value you enter is 1998, you receive an error. If you enter '03', Sybase IQ applies the default year and day and converts it to '1998-03-01'.
dd or DD jjj or JJJ	Represents number of day. Default day is 01. Always use leading zeros for number of day where appropriate, for example, '01' for first day. J or j indicates a Julian day (1 to 366) of the year.
hh HH	Represents hour. Hour is based on 24-hour clock. Always use leading zeros or blanks for hour where appropriate, for example, '01' for 1 am. '00' is also valid value for hour of 12 a.m.
nn	Represents minute. Always use leading zeros for minute where appropriate, for example, '08' for 8 minutes.
ss[.ssssss]	Represents seconds and fraction of a second.
aa	Represents the a.m. or p.m. designation.
pp	Represents the p.m designation only if needed. (This is an incompatibility with Sybase IQ versions earlier than 12.0; previously, “pp” was synonymous with “aa”.)
hh	Sybase IQ assumes zero for minutes and seconds. For example, if the DATETIME value you enter is '03', Sybase IQ converts it to '03:00:00.0000'.
hh:nn or hh:mm	Sybase IQ assumes zero for seconds. For example, if the time value you enter is '03:25', Sybase IQ converts it to '03:25:00.0000'.

Table 6-10: Sample DATE and DATETIME format options

Input data	Format specification
12/31/98	DATE ('MM/DD/YY')
19981231	DATE ('YYYYMMDD')
123198140150	DATETIME ('MMDDYYhhnnss')
14:01:50 12-31-98	DATETIME ('hh:mm:ss MM-DD-YY')
18:27:53	DATETIME ('hh:mm:ss')
12/31/98 02:01:50AM	DATETIME ('MM/DD/YY hh:mm:ssaa')

Sybase IQ has built-in load optimizations for common date, time, and datetime formats. If your data to be loaded matches one of these formats, you can significantly decrease load time by using the appropriate format. For a list of these formats, and details about optimizing performance when loading date and datetime data, see Chapter 7, “Moving Data In and Out of Databases” in the *Sybase IQ System Administration Guide*.

You can also specify the date/time field as an ASCII fixed-width field (as described above) and use the FILLER(1) option to skip the column delimiter. For more information about specifying date and time data, see Date and time data types on page 234 or Chapter 7, “Moving Data In and Out of Databases” in the *Sybase IQ System Administration Guide*.

The NULL portion of the *column-spec* 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 specifying a NULL value or reading a NULL value from the source file, the destination column must be able to contain NULLs.

ZEROS are interpreted as follows: the cell is set to NULL if (and only if) the input data (before conversion, if ASCII) is all binary zeros (and not character zeros).

- If the input data is character zero, then:
  a NULL (ZEROS) never causes the cell to be NULL.
  b NULL ('0') causes the cell to be NULL.
- If the input data is binary zero (all bits clear), then:
  a NULL (ZEROS) causes the cell to be NULL.
  b NULL ('0') never causes the cell to be NULL.

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

If the length of a VARCHAR cell is zero and the cell is not NULL, you get a zero-length cell. For all other data types, if the length of the cell is zero, Sybase IQ inserts a NULL. This is ANSI behavior. For non-ANSI treatment of zero-length character data, set the Non_Ansi_Null_Varchar database option.

Another important part of the *load-specification* is the FILLER option. It indicates you want to skip over a specified field in the source input file. For example, there may be characters at the end of rows or even entire fields in the input files that you do not want to add to the table. As with the *column-spec* definition, FILLER lets you specify ASCII fixed length of bytes, variable length characters delimited by a separator, and binary fields using PREFIX bytes.

filename-string The *filename-string* is passed to the server as a string. The string is therefore subject to the same formatting requirements as other SQL strings. In particular:

- To indicate directory paths in Windows systems, the backslash character \ must be represented by two backslashes. Therefore, the statement to load data from the file *c:\temp\input.dat* into the employee table is:

  ```
  LOAD TABLE employee
  FROM 'c:\\temp\\input.dat' ...
  ```

- The path name is relative to the database server, not to the client application. If you are running the statement on a database server on some other computer, the directory names refers to directories on the server machine, not on the client machine.

The following describes each of the clauses of the statement:

WORD SKIP Allows the load to continue when it encounters data longer than the limit specified when the word index was created.

If a row is not loaded because a word exceeds the maximum permitted size, a warning is written to the *.iqmsg* file. WORD size violations can be optionally logged to the MESSAGE LOG file and rejected rows logged to the ROW LOG file specified in the LOAD TABLE statement.

- If the option is not specified, LOAD TABLE reports an error and rolls back on the first occurrence of a word that is longer than the specified limit.

- *number* specifies the number of times the "Words exceeding the maximum permitted word length not supported" error is ignored.
- 0 (zero) means there is no limit.

QUOTES This parameter is optional and the default is ON. With QUOTES turned on, LOAD TABLE expects input strings to be enclosed in quote characters. The quote character is either an apostrophe (single quote) or a quotation mark (double quote). The first such character encountered in a string is treated as the quote character for the string. String data must be terminated with a matching quote.

With QUOTES ON, column or row delimiter characters can be included in the column value. Leading and ending quote characters are assumed not to be part of the value and are excluded from the loaded data value.

To include a quote character in a value with QUOTES ON, use two quotes. For example, the following line includes a value in the third column that is a single quote character:

```
'123 High Stree, Anytown', '(715)398-2354',''''
```

With STRIP turned on (the default), trailing blanks are stripped from values before they are inserted. Trailing blanks are stripped only for non-quoted strings. Quoted strings retain their trailing blanks. Leading blank or TAB characters are trimmed only when the QUOTES setting is ON.

The data extraction facility provides options for handling quotes (TEMP_EXTRACT_QUOTES, TEMP_EXTRACT_QUOTES_ALL, and TEMP_EXTRACT_QUOTE). If you plan to load back the extracted file with string fields which contain column or row delimiter under default ASCII extraction, use the TEMP_EXTRACT_BINARY option for the extract and the FORMAT 'binary' and QUOTES OFF options for LOAD TABLE.

Limits:

- The QUOTES ON option applies only to column-delimited ASCII fields.
- With QUOTES ON, the first character of a column delimiter or row terminator cannot be a single or double quote mark.
- The QUOTES option does not apply to loading binary large object (BLOB) or character large object (CLOB) data from the secondary file, regardless of its setting, A leading or trailing quote is loaded as part of CLOB data. Two consecutive quotes between enclosing quotes are loaded as two consecutive quotes with the QUOTES ON option.

- Adaptive Server Enterprise BCP does not support the QUOTES option. All field data is copied in or out equivalent to the QUOTES OFF setting. As QUOTES ON is the default setting for the Sybase IQ LOAD TABLE statement, you must specify QUOTES OFF when importing ASE data from BCP output to a Sybase IQ table.

Exceptions:

- If LOAD TABLE encounters any nonwhite characters after the ending quote character for an enclosed field, the following error is reported and the load operation is rolled back:

  ```
  Non-SPACE text found after ending quote character for
  an enclosed field.
  SQLSTATE: QTA14    SQLCODE: -1005014L
  ```

- With QUOTES ON, if a single or double quote is specified as the first character of the column delimiter, an error is reported and the load operation fails:

  ```
  Single or double quote mark cannot be the 1st character
  of column delimiter or row terminator with QUOTES option
  ON.
  SQLSTATE: QCA90    SQLCODE: -1013090L
  ```

For an example of the QUOTES option, see "Bulk loading data using the LOAD TABLE statement" in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

CHECK CONSTRAINTS This option defaults to ON. When you specify CHECK CONSTRAINTS ON, check constraints are evaluated and you are free to ignore or log them.

Setting CHECK CONSTRAINTS OFF causes Sybase IQ to ignore all check constraint violations. This can be useful, for example, during database rebuilding. If a table has check constraints that call user-defined functions that are not yet created, the rebuild fails unless this option is set to OFF.

This option is mutually exclusive to the following options. If any of these options are specified in the same load, an error results:

- IGNORE CONSTRAINT ALL
- IGNORE CONSTRAINT CHECK
- LOG ALL
- LOG CHECK

DEFAULTS If the DEFAULTS option is ON (the default) and the column has a default value, that value is used. If the DEFAULTS option is OFF, any column not present in the column list is assigned NULL.

The setting for the DEFAULTS option applies to all column DEFAULT values, including AUTOINCREMENT.

For detailed information on the use of column DEFAULT values with loads and inserts, see "Using column defaults" in Chapter 9, "Ensuring Data Integrity" in the *Sybase IQ System Administration Guide*.

ESCAPES If you omit a *column-spec* definition for an input field and ESCAPES is ON (the default), characters following the backslash character are recognized and interpreted as special characters by the database server. Newline characters can be included as the combination \n\, other characters can be included in data as hexadecimal ASCII codes, such as \x09 for the tab character. A sequence of two backslash characters (\\ ) is interpreted as a single backslash. For Sybase IQ, you must set this option OFF.

FORMAT Sybase IQ supports ASCII and binary input fields. The format is usually defined by the *column-spec* described above. If you omit that definition for a column, by default Sybase IQ uses the format defined by this option. Input lines are assumed to have ascii (the default) or binary fields, one row per line, with values separated by the column delimiter character.

DELIMITED BY If you omit a column delimiter in the column-spec definition, the default column delimiter character is a comma. You can specify an alternative column delimiter by providing a single ASCII character or the hexadecimal character representation. The DELIMITED BY clause is as follows:

```
... DELIMITED BY '\x09' ...
```

To use the newline character as a delimiter, you can specify either the special combination '\n' or its ASCII value '\x0a'. Although you can specify up to four characters in the column-spec *delimiter-string*, you can specify only a single character in the DELIMITED BY clause

STRIP With STRIP turned on (the default), trailing blanks are stripped from values before they are inserted. This is effective only for VARCHAR data; it does not apply to ASCII fix-width inserts. To turn the STRIP option off, the clause is as follows:

```
... STRIP OFF ...
```

Trailing blanks are stripped only for nonquoted strings. Quoted strings retain their trailing blanks. As an alternative, the FILLER option lets you be more specific in the number of bytes to strip instead of all the trailing spaces. It is more efficient for Sybase IQ to have this option off, and it adheres to the ANSI standard when dealing with trailing blanks. (char data is always padded, so this option only affects varchar data.)

WITH CHECKPOINT The default setting is OFF. If set to ON, a checkpoint is issued after successfully completing and logging the statement.

If WITH CHECKPOINT ON is not specified, and recovery is subsequently required, the data file used to load the table is needed for the recovery to complete successfully. If WITH CHECKPOINT ON is specified, and recovery is subsequently required, it begins after the checkpoint, and the data file need not be present.

BLOCK FACTOR Specifies blocking factor, or number of records per block used when a tape was created. This option is not valid for inserts from variable-length input fields; use the BLOCKSIZE option instead. However, it does affect all file inserts (including from disk) with fixed-length input fields, and it can dramatically affect performance. You cannot specify this option along with the BLOCK SIZE option. The default is 10,000.

BLOCK SIZE Specifies the default size in bytes in which input should be read. This option only affects variable length input data read from files; it is not valid for fixed length input fields. It is similar to BLOCK FACTOR, but there are no restrictions on the relationship of record size to block size. You cannot specify this option along with the BLOCK FACTOR option. The default is 500,000.

BYTE ORDER Specifies the byte order during reads. This option applies to all binary input fields. If none are defined, this option is ignored. Sybase IQ always reads binary data in the format native to the machine it is running on (default is NATIVE). You can also specify:

- HIGH when multibyte quantities have the high order byte first (for big endian platforms like Sun, IBM AIX, and HP).
- LOW when multibyte quantities have the low order byte first (for little endian platforms like Windows).

LIMIT Specifies the maximum number of rows to insert into the table. The default is 0 for no limit. The maximum is 2GB - 1.

NOTIFY Specifies that you be notified with a message each time the specified number of rows is successfully inserted into the table. The default is every 100,000 rows. The value of this option overrides the value of the NOTIFY_MODULUS database option.

ON FILE ERROR Specifies the action Sybase IQ takes when an input file cannot be opened because it does not exist or you have incorrect permissions to read the file. You can specify one of the following:

- ROLLBACK aborts the entire transaction (the default).
- FINISH finishes the insertions already completed and ends the load operation.
- CONTINUE returns an error but only skips the file to continue the load operation. You cannot use this option with partial-width inserts.

Only one ON FILE ERROR clause is permitted.

PREVIEW Displays the layout of input into the destination table including starting position, name, and data type of each column. Sybase IQ displays this information at the start of the load process. If you are writing to a log file, this information is also included in the log. This option is especially useful with partial-width inserts.

ROW DELIMITED BY Specifies a string up to 4 bytes in length that indicates the end of an input record. You can use this option only if all fields within the row are any of the following:

- Delimited with column terminators
- Data defined by the DATE or DATETIME *column-spec* options
- ASCII fixed length fields

You cannot use this option if any input fields contain binary data. With this option, a row terminator causes any missing fields to be set to NULL. All rows must have the same row delimiters, and it must be distinct from all column delimiters. The row and field delimiter strings cannot be an initial subset of each other. For example, you cannot specify “*” as a field delimiter and “*#” as the row delimiter, but you could specify “#” as the field delimiter with that row delimiter.

If a row is missing its delimiters, Sybase IQ returns an error and rolls back the entire load transaction. The only exception is the final record of a file where it rolls back that row and returns a warning message. On Windows, a row delimiter is usually indicated by the newline character followed by the carriage return character. You might need to specify this as the *delimiter-string* (see above for description) for either this option or FILLER.

SKIP Lets you define a number of rows to skip at the beginning of the input tables for this load. The default is 0.

START ROW ID Specifies the record identification number of a row in the Sybase IQ table where it should start inserting. This option is used for *partial-width* inserts, which are inserts into a subset of the columns in the table. By default, new rows are inserted wherever there is space in the table, and each insert starts a new row. Partial-width inserts need to start at an existing row. They also need to insert data from the source file into the destination table positionally by column, so you must specify the destination columns in the same order as their corresponding source columns. Define the format of each input column with a *column-spec*. The default is 0. For more information about partial-width inserts see Chapter 7, “Moving Data In and Out of Databases” in the *Sybase IQ System Administration Guide*.

Use the START ROW ID option for partial-width inserts only. If the columns being loaded already contain data, the insert fails.

UNLOAD FORMAT Specifies that the file has Sybase IQ internal unload formats for each column created by an earlier version of Sybase IQ (before Version 12.0). This load option has the following restrictions:

- You cannot specify any *column-spec* (such as ASCII or PREFIX) for a column other than BINARY. This includes the NULL specifications.
- If you need to load null values for a column using the BINARY *column-spec*, you must specify the WITH NULL BYTE keyword or Sybase IQ returns an error.
- You cannot use the DELIMITED BY or ROW DELIMITED BY options with UNLOAD FORMAT.

ON PARTIAL INPUT ROW Specifies the action to take when a partial input row is encountered during a load. You can specify one of the following:

- CONTINUE issues a warning and continues the load operation. This is the default.
- ROLLBACK aborts the entire load operation and reports the error.

```
Partial input record skipped at EOF.
SQLSTATE: QDC32    SQLSTATE: -1000232L
```

IGNORE CONSTRAINT Specifies whether to ignore CHECK, UNIQUE, NULL, DATA VALUE, and FOREIGN KEY integrity constraint violations that occur during a load and the maximum number of violations to ignore before initiating a rollback. Specifying each *constrainttype* has the following result:

- **CHECK *limit*** If *limit* specifies zero, the number of UNIQUE constraint violations to ignore is infinite. If CHECK is not specified, the first occurrence of any CHECK constraint violation causes the LOAD statement to roll back. If *limit* is nonzero, then the *limit* +1 occurrence of a CHECK constraint violation causes the load to roll back.
- **UNIQUE *limit*** If *limit* specifies zero, then the number of UNIQUE constraint violations to ignore is infinite. If *limit* is nonzero, then the *limit* +1 occurrence of a UNIQUE constraint violation causes the load to roll back.
- **NULL *limit*** If *limit* specifies zero, then the number of NULL constraint violations to ignore is infinite. If *limit* is nonzero, then the *limit* +1 occurrence of a NULL constraint violation causes the load to roll back.
- **FOREIGN KEY *limit*** If *limit* specifies zero, the number of FOREIGN KEY constraint violations to ignore is infinite. If *limit* is nonzero, then the *limit* +1 occurrence of a FOREIGN KEY constraint violation causes the load to roll back.
- **DATA VALUE *limit*** If the database option CONVERSION_ERROR = ON, an error is reported and the statement rolls back. If *limit* specifies zero, then the number of DATA VALUE constraint violations (data type conversion errors) to ignore is infinite. If *limit* is nonzero, then the *limit* +1 occurrence of a DATA VALUE constraint violation causes the load to roll back.
- **ALL *limit*** If the database option CONVERSION_ERROR = ON, an error is reported and the statement rolls back. If *limit* specifies zero, then the cumulative total of all integrity constraint violations to ignore is infinite. If *limit* is nonzero, then load rolls back when the cumulative total of all ignored UNIQUE, NULL, DATA VALUE, and FOREIGN KEY integrity constraint violations exceeds the value of *limit*. For example, you specify the following IGNORE CONSTRAINT option:

```
IGNORE CONSTRAINT NULL 50, UNIQUE 100, ALL 200
```

The total number of integrity constraint violations cannot exceed 200, whereas the total number of NULL and UNIQUE constraint violations cannot exceed 50 and 100, respectively. Whenever any of these limits is exceeded, the LOAD TABLE statement rolls back.

Note A single row can have more than one integrity constraint violation. Every occurrence of an integrity constraint violation counts towards the limit of that type of violation.

Sybase strongly recommends setting the IGNORE CONSTRAINT option limit to a nonzero value if you are logging the ignored integrity constraint violations. Logging an excessive number of violations affects the performance of the load.

If CHECK, UNIQUE, NULL, or FOREIGN KEY is not specified in the IGNORE CONSTRAINT clause, then the load rolls back on the first occurrence of each of these types of integrity constraint violation.

If DATA VALUE is not specified in the IGNORE CONSTRAINT clause, then the load rolls back on the first occurrence of this type of integrity constraint violation, unless the database option CONVERSION_ERROR = OFF. If CONVERSION_ERROR = OFF, a warning is reported for any DATA VALUE constraint violation and the load continues.

When the load completes, an informational message regarding integrity constraint violations is logged in the *.iqmsg* file. This message contains the number of integrity constraint violations that occurred during the load and the number of rows that were skipped.

MESSAGE LOG Specifies the names of files in which to log information about integrity constraint violations and the types of violations to log. Timestamps indicating the start and completion of the load are logged in both the MESSAGE LOG and the ROW LOG files. Both MESSAGE LOG and ROW LOG must be specified, or no information about integrity violations is logged.

- If the ONLY LOG clause is not specified, no information on integrity constraint violations is logged. Only the timestamps indicating the start and completion of the load are logged.
- Information is logged on all integrity constraint-type violations specified in the ONLY LOG clause or for all word index-length violations if the keyword WORD is specified.
- If constraint violations are being logged, every occurrence of an integrity constraint violation generates exactly one row of information in the MESSAGE LOG file.

The number of rows (errors reported) in the MESSAGE LOG file can exceed the IGNORE CONSTRAINT option limit, because the load is performed by multiple threads running in parallel. More than one thread might report that the number of constraint violations has exceeded the specified limit.

- If constraint violations are being logged, exactly one row of information is logged in the ROW LOG file for a given row, regardless of the number of integrity constraint violations that occur on that row.

  The number of distinct errors in the MESSAGE LOG file might not exactly match the number of rows in the ROW LOG file. The difference in the number of rows is due to the parallel processing of the load described above for the MESSAGE LOG.

- The MESSAGE LOG and ROW LOG files cannot be raw partitions.
- If the MESSAGE LOG or ROW LOG file already exists, new information is appended to the file.
- Specifying an invalid file name for the MESSAGE LOG or ROW LOG file generates an error.
- Specifying the same file name for the MESSAGE LOG and ROW LOG files generates an error.

Various combinations of the IGNORE CONSTRAINT and MESSAGE LOG options result in different logging actions, as indicated in Table 6-11.

Table 6-11: LOAD TABLE logging actions

IGNORE CONSTRAINT specified?	MESSAGE LOG specified?	Action
yes	yes	All ignored integrity constraint violations are logged, including the user specified limit, before the rollback.
no	yes	The first integrity constraint violation is logged before the rollback.
yes	no	Nothing is logged.
no	no	Nothing is logged. The first integrity constraint violation causes a rollback.

Note Sybase strongly recommends setting the IGNORE CONSTRAINT option limit to a nonzero value, if you are logging the ignored integrity constraint violations. If a single row has more than one integrity constraint violation, a row for *each* violation is written to the MESSAGE LOG file. Logging an excessive number of violations affects the performance of the load.

LOG DELIMITED BY Specifies the separator between data values in the ROW LOG file. The default separator is a comma.

For more details on the contents and format of the MESSAGE LOG and ROW LOG files, see "Bulk loading data using the LOAD TABLE statement" in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*.

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions

The permissions required to execute a LOAD TABLE statement depend on the database server -gl command line option, as follows:

- If the -gl option is set to ALL, you must be the owner of the table, have DBA authority, or have ALTER permission.
- If the -gl option is set to DBA, you must have DBA authority.
- If the -gl option is set to NONE, LOAD TABLE is not permitted.

For more information, see the -gl command line option in "Server command-line switches" on page 8 in Chapter 1, "Running the Database Server" in the *Sybase IQ Utility Guide*.

LOAD TABLE also requires an exclusive lock on the table.

See also

INSERT statement on page 568

"LOAD_ZEROLENGTH_ASNULL option" on page 103

"NON_ANSI_NULL_VARCHAR option" on page 124

"Bulk loading data using the LOAD TABLE statement" in Chapter 7, "Moving Data In and Out of Databases" in the *Sybase IQ System Administration Guide*

"Monitoring disk space usage," Chapter 1, "Troubleshooting Hints," in the *Sybase IQ Troubleshooting and Recovery Guide*

LOCK TABLE statement

Description

Prevents other concurrent transactions from accessing or modifying a table.

Syntax

LOCK TABLE *table-name* [**WITH HOLD**] **IN** { **SHARE** | **EXCLUSIVE** } **MODE**

Examples

Prevents other transactions from modifying the customer table for the duration of the current transaction:

```
LOCK TABLE customer IN SHARE MODE
```

Usage

table-name The table must be a base table, not a view. As temporary table data is local to the current connection, locking global or local temporary tables has no effect.

WITH HOLD If this clause is specified, the lock is held until the end of the connection. If the clause is not specified, the lock is released when the current transaction is committed or rolled back.

SHARE Prevents other transactions from modifying the table, but allows them read access. In this mode, you can change data in the table as long as no other transaction has locked the row being modified, either indirectly, or explicitly by using LOCK TABLE.

EXCLUSIVE Prevents other transactions from accessing the table. In this mode, no other transaction can execute queries, updates of any kind, or any other action against the table. If a table t is locked exclusively with LOCK TABLE *t* IN EXCLUSIVE MODE, the default server behavior is to not acquire row locks for *t*. This behavior can be disabled by setting the SUBSUME_ROW_LOCKS option OFF.

The LOCK TABLE statement allows direct control over concurrency at a table level, independent of the current isolation level.

Whereas the isolation level of a transaction generally governs the kinds of locks that are set when the current transaction executes a request, the LOCK TABLE statement allows more explicit control locking of the rows in a table.

The locks placed by LOCK TABLE in SHARE mode are phantom and anti-phantom locks, which are displayed by the sa_locks procedure as PT and AT.

Standards

- **SQL92** Vendor extension.
- **Sybase** Supported in Adaptive Server Enterprise. The WITH HOLD clause is not supported in Adaptive Server Enterprise. Adaptive Server Enterprise provides a WAIT clause that is not supported in Adaptive Server Anywhere.

Permissions

To lock a table in SHARE mode, SELECT privileges are required.

To lock a table in EXCLUSIVE mode; you must be the table owner or have DBA authority.

See also

SELECT statement on page 632

LOOP statement

Description

Repeats the execution of a statement list.

Syntax

[*statement-label*:]
... [**WHILE** *search-condition*] **LOOP**
... *statement-list*
... **END LOOP** [*statement-label*]

Examples

- A WHILE loop in a procedure:

```
...
SET i = 1 ;
WHILE i <= 10 LOOP
    INSERT INTO Counters( number ) VALUES ( i ) ;
```

```
    SET i = i + 1 ;
END LOOP ;
...
```

- A labeled loop in a procedure:

```
SET i = 1;
lbl:
LOOP
    INSERT
    INTO Counters( number )
    VALUES ( i ) ;
    IF i >= 10 THEN
       LEAVE lbl ;
    END IF ;
    SET i = i + 1 ;
END LOOP lbl
```

Usage

The WHILE and LOOP statements are control statements that let you repeatedly execute a list of SQL statements while a *search-condition* evaluates to TRUE. The LEAVE statement can be used to resume execution at the first statement after the END LOOP.

If the ending *statement-label* is specified, it must match the beginning *statement-label*.

Side effects

None.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** Not supported in Adaptive Server Enterprise. The WHILE statement provides looping in Transact-SQL stored procedures.

Permissions

None.

See also

FOR statement on page 551

LEAVE statement on page 578

MESSAGE statement

Description

Displays a message.

Syntax

MESSAGE *expression*, ...
[**TYPE** { **INFO** | **ACTION** | **WARNING** | **STATUS** }]
[**TO** {**CONSOLE** | **CLIENT** [**FOR** { **CONNECTION** *conn_id* | **ALL** }] | **LOG** }
[**DEBUG ONLY**]]

conn_id : integer

Parameters

TYPE The TYPE clause has an effect only if the message is sent to the client. The client application must decide how to handle the message. Interactive SQL displays messages in the following locations:

- **INFO** – The Message window (default).
- **ACTION**– A Message box with an OK button.
- **WARNING** – A Message box with an OK button.
- **STATUS** – The Messages pane.

TO Specifies the destination of a message:

- **CONSOLE** – Send messages to the database server window. CONSOLE is the default.
- **CLIENT** – Send messages to the client application. Your application must decide how to handle the message, and you can use the TYPE as information on which to base that decision.
- **12.jar** – Send messages to the server log file specified by the -o option.

FOR For messages TO CLIENT, this clause specifies which connections receive notification about the message:

- **CONNECTION conn_id** – Specifies the recipient's connection ID for the message.
- **ALL** – Specifies that all open connections receive the message.

DEBUG ONLY Lets you control whether debugging messages added to stored procedures are enabled or disabled by changing the setting of the DEBUG_MESSAGES option. When DEBUG ONLY is specified, the MESSAGE statement is executed only when the DEBUG_MESSAGES option is set to ON.

Note DEBUG ONLY messages are inexpensive when the DEBUG_MESSAGES option is set to OFF, so these statements can usually be left in stored procedures on a production system. However, they should be used sparingly in locations where they would be executed frequently; otherwise, they might result in a small performance penalty.

Examples

- Displays a message on the server message window:

```
CREATE PROCEDURE message_test ()
BEGIN
MESSAGE 'The current date and time: ', Now();
END
```

- Displays the string `The current date and time,` and the current date and time, on the database server message window:

```
CALL message_test()
```

- To register a callback in ODBC, first declare the message handler:

```
void SQL_CALLBACK my_msgproc(
    void *    sqlca,
    unsigned char     msg_type,
    long              code,
    unsigned short    len,
    char*              msg )
{ … }
```

Install the declared message handler by calling the SQLSetConnectAttr function.

```
rc = SQLSetConnectAttr(
    dbc,
    ASA_REGISTER_MESSAGE_CALLBACK,
    (SQLPOINTER) &my_msgproc, SQL_IS_POINTER );
```

Usage

The MESSAGE statement displays a message, which can be any expression. Clauses can specify where the message is displayed.

The procedure issuing a MESSAGE … TO CLIENT statement must be associated with a connection.

For example, the message box is not displayed in the following example because the event occurs outside of a connection.

```
CREATE EVENT CheckIdleTime TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
   MESSAGE 'Idle engine' type warning to client;
END;
```

However, in the following example, the message is written to the server console.

```
CREATE EVENT CheckIdleTime TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
   MESSAGE 'Idle engine' type warning to console;
END;
```

Valid expressions can include a quoted string or other constant, variable, or function. However, queries are not permitted in the output of a MESSAGE statement even though the definition of an expression includes queries.

The FOR clause can be used to notify another application of an event detected on the server without the need for the application to explicitly check for the event. When the FOR clause is used, recipients receive the message the next time that they execute a SQL statement. If the recipient is currently executing a SQL statement, the message is received when the statement completes. If the statement being executed is a stored procedure call, the message is received before the call is completed.

If an application requires notification within a short time after the message is sent and when the connection is not executing SQL statements, you can use a second connection. This connection can execute one or more WAITFOR DELAY statements. These statements do not consume significant resources on the server or network (as would happen with a polling approach), but permit applications to receive notification of the message shortly after it is sent.

ESQL and ODBC clients receive messages via message callback functions. In each case, these functions must be registered. To register ESQL message handlers, use the db_register_callback function.

ODBC clients can register callback functions using the SQLSetConnectAttr function.

Side effects

None.

Standards

- **SQL92** Vendor extension.

- **SQL99** Vendor extension.
- **Sybase** Not supported in Adaptive Server Enterprise. The Transact-SQL PRINT statement provides a similar feature, and is available in Adaptive Server Anywhere.

Permissions

Must be connected to the database.

DBA authority is required to execute a MESSAGE statement containing a FOR clause.

See also

CREATE PROCEDURE statement on page 485

"DEBUG_MESSAGES option" on page 68

Adaptive Server Anywhere Programming Guide for information about using callback functions

OPEN statement [ESQL] [SP]

Description

Opens a previously declared cursor to access information from the database.

Syntax

OPEN *cursor-name*
... [**USING** [**DESCRIPTOR** { *sqlda-name* | *host-variable* [, ...] }]]
... [**WITH HOLD**]

Parameters

cursor-name:
identifier or host-variable

sqlda-name:
identifier

Examples

- Examples showing the use of OPEN in Embedded SQL:

```
1. EXEC SQL OPEN employee_cursor;
2. EXEC SQL PREPARE emp_stat FROM
'SELECT empnum, empname FROM employee WHERE name like
?';
EXEC SQL DECLARE employee_cursor CURSOR FOR emp_stat;
EXEC SQL OPEN employee_cursor USING :pattern;
```

- An example from a procedure:

```
BEGIN
DECLARE cur_employee CURSOR FOR
   SELECT emp_lname
   FROM employee ;
DECLARE name CHAR(40) ;
```

```
OPEN cur_employee;
LOOP
FETCH NEXT cur_employee into name ;
     ...
END LOOP
CLOSE cur_employee;
END
```

Usage

By default, all cursors are automatically closed at the end of the current transaction (COMMIT or ROLLBACK). The optional WITH HOLD clause keeps the cursor open for subsequent transactions. The cursor remains open until the end of the current connection or until an explicit CLOSE statement is executed. Cursors are automatically closed when a connection is terminated.

The cursor is positioned before the first row . See Chapter 8, "Using Procedures and Batches" of the *Sybase IQ System Administration Guide*.

Embedded SQL

The USING DESCRIPTOR sqlda-name, host-variable and BLOCK n formats are for Embedded SQL only.

If the cursor name is specified by an identifier or string, then the corresponding DECLARE CURSOR statement must appear prior to the OPEN in the C program; if the cursor name is specified by a host variable, then the DECLARE CURSOR statement must execute before the OPEN statement.

The optional USING clause specifies the host variables that are bound to the placeholder bind variables in the SELECT statement for which the cursor has been declared.

After successful execution of the OPEN statement, the *sqlerrd[3]* field of the SQLCA (SQLIOESTIMATE) is filled in with an estimate of the number of input/output operations required to fetch all rows of the query. Also, the *sqlerrd[2]* field of the SQLCA (SQLCOUNT) is filled in with either the actual number of rows in the cursor (a value greater than or equal to 0), or an estimate thereof (a negative number whose absolute value is the estimate). The *sqlerrd[2]* field is the actual number of rows, if the database server can compute this value without counting the rows.

Side effects

None.

Standards

- **SQL92** Embedded SQL use is an entry-level feature. Use of procedures is a Persistent Stored Module feature.

- **Sybase** The simple OPEN *cursor-name* syntax is supported by Adaptive Server Enterprise. None of the other clauses are supported in Adaptive Server Enterprise stored procedures. Open Client/Open Server supports the USING descriptor or host name variable syntax.

Permissions

- Must have SELECT permission on all tables in a SELECT statement or EXECUTE permission on the procedure in a CALL statement.
- When the cursor is on a CALL statement, OPEN causes the procedure to execute until the first result set (SELECT statement with no INTO clause) is encountered. If the procedure completes and no result set is found, the SQLSTATE_PROCEDURE_COMPLETE warning is set.

See also

CLOSE statement [ESQL] [SP] on page 434

DECLARE CURSOR statement [ESQL] [SP] on page 516

FETCH statement [ESQL] [SP] on page 547

PREPARE statement [ESQL] on page 611

RESUME statement on page 626

OUTPUT statement [DBISQL]

Description

Writes the current query results to a file.

Syntax

OUTPUT TO *filename*
[**APPEND**] [**VERBOSE**]
[**FORMAT** *output-format*]
[**ESCAPE CHARACTER** *character*]
[**DELIMITED BY** *string*]
[**QUOTE** *string* [**ALL**]]
[**COLUMN WIDTHS (***integer*, …**)**]
[**HEXADECIMAL** { **ON** | **OFF** | **ASIS** }]
[**ENCODING** *encoding*]

Parameters

output-format:
ASCII| DBASEII | DBASEIII | EXCEL | FIXED | FOXPRO | HTML | LOTUS | SQL | XML

encoding:
string or *identifier*

Examples

Example 1 Places the contents of the employee table in a file in ASCII format:

```
SELECT * FROM employee; OUTPUT TO employee.txt FORMAT
ASCII
```

Example 2 Places the contents of the employee table at the end of an existing file, and includes any messages about the query in this file as well:

```
SELECT * FROM employee; OUTPUT TO employee.txt APPEND
VERBOSE
```

Example 3 Suppose you need to export a value that contains an embedded line feed character. A line feed character has the numeric value 10, which you can represent as the string '\x0a' in a SQL statement. You could execute the following statement, with HEXADECIMAL ON:

```
SELECT 'line1\x0aline2'; OUTPUT TO file.txt HEXADECIMAL
ON
```

You get a file with one line in it, containing the following text:

```
line10x0aline2
```

If you execute the same statement with HEXADEMICAL OFF, you get the following:

```
line1\x0aline2
```

Finally, if you set HEXADECIMAL to ASIS, you get a file with two lines:

```
line1 line2
```

Using ASIS generates two lines because the embedded line feed character has been exported without being converted to a two-digit hex representation, and without a prefix.

Usage

The OUTPUT statement copies the information retrieved by the current query to a file.

You can specify the output format with the optional FORMAT clause. If no FORMAT clause is specified, the Interactive SQL OUTPUT_FORMAT option setting is used.

The current query is the SELECT or LOAD TABLE statement that generated the information that appears on the Results tab in the Results pane. The OUTPUT statement reports an error if there is no current query.

Note OUTPUT is especially useful in making the results of a query or report available to another application, but it is not recommended for bulk operations. For high-volume data movement, use the ASCII and BINARY data extraction functionality with the SELECT statement. The extraction functionality provides much better performance for large-scale data movement, and creates an output file you can use for loads.

APPEND This optional keyword is used to append the results of the query to the end of an existing output file without overwriting the previous contents of the file. If the APPEND clause is not used, the OUTPUT statement overwrites the contents of the output file by default. The APPEND keyword is valid if the output format is ASCII, FIXED, or SQL.

VERBOSE When the optional VERBOSE keyword is included, error messages about the query, the SQL statement used to select the data, and the data itself are written to the output file. If VERBOSE is omitted (the default), only the data is written to the file. The VERBOSE keyword is valid if the output format is ASCII, FIXED, or SQL.

FORMAT Allowable output formats are:

- **ASCII** The output is an ASCII format file with one row per line in the file. All values are separated by commas, and strings are enclosed in apostrophes (single quotes). The delimiter and quote strings can be changed using the DELIMITED BY and QUOTE clauses. If ALL is specified in the QUOTE clause, all values (not just strings) are quoted.

  Three other special sequences are also used. The two characters \n represent a newline character, \\ represents a single \, and the sequence \xDD represents the character with hexadecimal code DD. This is the default output format.

  If you are exporting Java methods that have string return values, you must use the HEXADECIMAL OFF clause.

- **DBASEII** The output is a dBASE II format file with the column definitions at the top of the file. Note that a maximum of 32 columns can be output. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.

- **DBASEIII** The output is a dBASE III format file with the column definitions at the top of the file. Note that a maximum of 128 columns can be output. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.

- **EXCEL** The output is an Excel 2.1 worksheet. The first row of the worksheet contains column labels (or names, if there are no labels defined). Subsequent worksheet rows contain the actual table data.

- **FIXED** The output is fixed format with each column having a fixed width. The width for each column can be specified using the COLUMN WIDTHS clause. No column headings are output in this format.

If COLUMN WIDTHS is omitted, the width for each column is computed from the data type for the column, and is large enough to hold any value of that data type. The exception is that LONG VARCHAR and LONG BINARY data defaults to 32KB.

- **FOXPRO** The output is a FoxPro format file (the FoxPro memo field is different than the dBASE memo field) with the column definitions at the top of the file. Note that a maximum of 128 columns can be output. Column names are truncated to 11 characters. Column names are truncated to 11 characters, and each row of data in each column is truncated to 255 characters.
- **HTML** The output is in the Hyper Text Markup Language format.
- **LOTUS** The output is a Lotus WKS format worksheet. Column names are put as the first row in the worksheet. Note that there are certain restrictions on the maximum size of Lotus WKS format worksheets that other software (such as Lotus 1-2-3) can load. There is no limit to the size of file Interactive SQL can produce.
- **SQL** The output is an Interactive SQL INPUT statement required to recreate the information in the table.

  Note Sybase IQ does not support the INPUT statement. You would need to edit this statement to a valid LOAD TABLE (or INSERT) statement to use it to load data back in.

- **XML** The output is an XML file encoded in UTF-8 and containing an embedded DTD. Binary values are encoded in CDATA blocks with the binary data rendered as 2-hex-digit strings. The LOAD TABLE statement does not accept XML as a file format.

ESCAPE CHARACTER The default escape character for characters stored as hexadecimal codes and symbols is a backslash (\), so \x0A is the linefeed character, for example.

This default can be changed using the ESCAPE CHARACTER clause.
For example, to use the exclamation mark as the escape character, you would enter:

```
... ESCAPE CHARACTER '!'
```

DELIMITED BY The DELIMITED BY clause is for the ASCII output format only. The delimiter string is placed between columns (default comma).

QUOTE The QUOTE clause is for the ASCII output format only. The quote string is placed around string values. The default is a single quote character. If ALL is specified in the QUOTE clause, the quote string is placed around all values, not just around strings.

COLUMN WIDTHS The COLUMN WIDTHS clause is used to specify the column widths for the FIXED format output.

HEXADECIMAL The HEXADECIMAL clause specifies how binary data is to be unloaded for the ASCII format only. When set to ON, binary data is unloaded in the format 0xabcd. When set to OFF, binary data is escaped when unloaded (\xab\xcd). When set to ASIS, values are written as is, that is, 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.

ENCODING The *encoding* argument lets you specify the encoding that is used to write the file. The ENCODING clause can be used only with the ASCII format.

If *encoding* is not specified, Interactive SQL determines the code page that is used to write the file as follows, where code page values occurring earlier in the list take precedence over those occurring later:

- The code page specified with the DEFAULT_ISQL_ENCODING option (if this option is set)
- The code page specified with the -codepage option when Interactive SQL was started
- The default code page for the computer Interactive SQL is running on

Side effects

In Interactive SQL, the Results tab displays only the results of the current query. All previous query results are replaced with the current query results.

Standards

- **SQL92** Vendor extension.
- **SQL99** Vendor extension.
- **Sybase** Not applicable.

Permissions

None

See also

DEFAULT_ISQL_ENCODING option [DBISQL] on page 69

OUTPUT_FORMAT option [ISQL] on page 130

SELECT statement on page 632

PARAMETERS statement [DBISQL]

Description

Specifies parameters to a DBISQL command file.

Syntax

PARAMETERS *parameter1*, *parameter2*, ...

Examples

The following DBISQL command file takes two parameters:

```
PARAMETERS department_id, file ;
SELECT emp_lname
FROM employee
WHERE dept_id = {department_id}
>#{file}.dat;
```

Usage

PARAMETERS specifies how many parameters there are to a command file and also names those parameters so that they can be referenced later in the command file.

Parameters are referenced by putting into the file where you want the named parameter to be substituted.:

```
{parameter1}
```

There must be no spaces between the braces and the parameter name.

If a command file is invoked with fewer than the required number of parameters, DBISQL prompts for values of the missing parameters.

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions

None

See also

READ statement [DBISQL] on page 617

PREPARE statement [ESQL]

Description

Prepares a statement to be executed later or used for a cursor.

Syntax

PREPARE *statement-name*
FROM *statement*
... [**DESCRIBE** *describe-type* **INTO** [[**SQL**] **DESCRIPTOR**] *descriptor*]
... [**WITH EXECUTE**]

Parameters

statement-name:
identifier or host-variable

statement:
string, or host-variable

describe-type:
{ ALL | BIND VARIABLES | INPUT | OUTPUT | SELECT LIST }
... { LONG NAMES [[OWNER.]TABLE.]COLUMN] | WITH VARIABLE RESULT }

Examples

Prepares a simple query:

```
EXEC SQL PREPARE employee_statement FROM
'SELECT emp_lname FROM employee';
```

Usage

The PREPARE statement prepares a SQL statement from the *statement* and associates the prepared statement with *statement-name*. This statement name is referenced to execute the statement, or to open a cursor if the statement is a SELECT statement. *Statement-name* may be a host variable of type a_sql_statement_number defined in the sqlca.h header file that is automatically included. If an identifier is used for the *statement-name*, only one statement per module may be prepared with this *statement-name*.

If a host variable is used for *statement-name*, it must have the type short int. There is a typedef for this type in sqlca.h called a_sql_statement_number. This type is recognized by the SQL preprocessor and can be used in a DECLARE section. The host variable is filled in by the database during the PREPARE statement and need not be initialized by the programmer.

If the DESCRIBE INTO DESCRIPTOR clause is used, the prepared statement is described into the specified descriptor. The describe type may be any of the describe types allowed in the DESCRIBE statement.

If the WITH EXECUTE clause is used, the statement is executed if and only if it is not a CALL or SELECT statement, and it has no host variables. The statement is immediately dropped after a successful execution. If PREPARE and DESCRIBE (if any) are successful but the statement cannot be executed, a warning SQLCODE 111, SQLSTATE 01W08 is set, and the statement is not dropped.

The DESCRIBE INTO DESCRIPTOR and WITH EXECUTE clauses might improve performance, as they decrease the required client/server communication.

Describing variable result sets

The WITH VARIABLE RESULT clause is used to describe procedures that may have more than one result set, with different numbers or types of columns.

If WITH VARIABLE RESULT is used, the database server sets the SQLCOUNT value after the describe to one of the following values:

- **0** The result set may change: the procedure call should be described again following each OPEN statement.
- **1** The result set is fixed. No redescribing is required.

Statements that can be prepared

The following is a list of statements that can be PREPARED:

- ALTER
- CALL
- COMMENT ON
- CREATE
- DELETE
- DROP
- GRANT
- INSERT
- REVOKE
- SELECT
- SET OPTION

Compatibility issue

For compatibility reasons, preparing COMMIT, PREPARE TO COMMIT, and ROLLBACK statements is still supported. However, we recommend that you do all transaction management operations with static Embedded SQL because certain application environments may require it. Also, other Embedded SQL systems do not support dynamic transaction management operations.

Note Make sure that you DROP the statement after use. If you do not, then the memory associated with the statement is not reclaimed.

Side effects

Any statement previously prepared with the same name is lost.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions

None.

See also

DECLARE CURSOR statement [ESQL] [SP] on page 516

DESCRIBE statement [ESQL] on page 528

DROP STATEMENT statement [ESQL] on page 539

EXECUTE statement [ESQL] on page 541

OPEN statement [ESQL] [SP] on page 603

PRINT statement [T-SQL]

Description

Displays a message on the message window of the database server.

Syntax

PRINT *format-string* [, *arg-list*]

Examples

Example 1 Displays a message on the server message window:

```
CREATE PROCEDURE print_test
AS
PRINT 'Procedure called successfully'
```

This statement returns the string “Procedure called successfully” to the client:

```
EXECUTE print_test
```

Example 2 Illustrates the use of placeholders in the PRINT statement:

```
DECLARE @var1 INT, @var2 INT
SELECT @var1 = 3, @var2 = 5
PRINT 'Variable 1 = %1!, Variable 2 = %2!', @var1, @var2
```

Example 3 Uses RAISERROR to disallow connections:

```
CREATE procedure DBA.login_check()
begin
    // Allow a maximum of 3 concurrent connections
    IF( db_property('ConnCount') > 3 ) then
    raiserror 28000
        'User %1! is not allowed to connect -- there are
           already %2! users logged on',
           current user,
           cast(db_property('ConnCount') as int)-1;
        ELSE
    call sp_login_environment;
        end if;
end
go
grant execute on DBA.login_check to PUBLIC
go
set option PUBLIC.Login_procedure='DBA.login_check'
go
```

For an alternate way to disallow connections, see "LOGIN_PROCEDURE option" on page 106 or "sp_iqmodifylogin procedure" on page 809.

Usage

The PRINT statement returns a message to the client window if you are connected from an Open Client application or JDBC application. If you are connected from an Embedded SQL or ODBC application, the message displays on the database server window.

The format string can contain placeholders for the arguments in the optional argument list. These placeholders are of the form *%nn!*, where *nn* is an integer between 1 and 20.

Side effects

None.

Standards

- **SQL92** Transact-SQL extension.
- **Sybase** Supported by Adaptive Server Enterprise.

Permissions

Must be connected to the database.

See also

MESSAGE statement on page 600

PUT statement [ESQL]

Description

Inserts a row into the specified cursor.

Syntax

PUT *cursor-name* [**USING DESCRIPTOR** *sqlda-name* | **FROM** *hostvar-list*] [**INTO** { **DESCRIPTOR** *into-sqlda-name* | *into-hostvar-list* }] [**ARRAY** *:nnn*]

cursor-name : *identifier* or *hostvar*

sqlda-name : *identifier*

hostvar-list : may contain indicator variables

Examples

The following statement illustrates the use of PUT in Embedded SQL:

```
EXEC SQL PUT cur_employee FROM :emp_id, :emp_lname;
```

Usage

Inserts a row into the named cursor. Values for the columns are taken from the first SQLDA or the host variable list, in a one-to-one correspondence with the columns in the INSERT statement (for an INSERT cursor) or the columns in the select list (for a SELECT cursor).

The PUT statement can be used only on a cursor over an INSERT or SELECT statement that references a single table in the FROM clause, or that references an updatable view consisting of a single base table.

If the sqldata pointer in the SQLDA is the null pointer, no value is specified for that column. If the column has a DEFAULT VALUE associated with it, that is used; otherwise, a NULL value is used.

The second SQLDA or host variable list contains the results of the PUT statement.

The optional ARRAY clause can be used to carry out wide puts, which insert more than one row at a time and which might improve performance. The value nnn is the number of rows to be inserted. The SQLDA must contain nnn * (columns per row) variables. The first row is placed in SQLDA variables 0 to (columns per row)-1, and so on.

Inserting into a cursor
For scroll (values-sensitive) cursors, the inserted row appears if the new row matches the WHERE clause and the keyset cursor has not finished populating. For dynamic cursors, if the inserted row matches the WHERE clause, the row might appear. Insensitive cursors cannot be updated.

For information on putting LONG VARCHAR or LONG BINARY values into the database, see .

Side Effects

When inserting rows into a value-sensitive (keyset-driven) cursor, the inserted rows appear at the end of the result set, even when they do not match the WHERE clause of the query or if an ORDER BY clause would normally have placed them at another location in the result set. For more information, see the *Adaptive Server Anywhere Programming Guide*.

Standards

- **SQL92** Entry-level feature.
- **SQL99** Core feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions

Must have INSERT permission.

See also

DELETE (positioned) statement [ESQL] [SP] on page 527

INSERT statement on page 568

UPDATE statement on page 661

UPDATE (positioned) statement [ESQL] [SP] on page 664

RAISERROR statement [T-SQL]

Description

Signals an error and sends a message to the client.

Syntax

RAISERROR *error-number* [*format-string*] [, *arg-list*]

Examples

Raises error 99999, which is in the range for user-defined errors, and sends a message to the client:

```
RAISERROR 99999 'Invalid entry for this
column: %1!', @val
```

There is no comma between the *error-number* and the *format-string* parameters. The first item following a comma is interpreted as the first item in the argument list.

Usage

The RAISERROR statement allows user-defined errors to be signaled, and sends a message on the client.

The *error-number* is a 5-digit integer greater than 17000. The error number is stored in the global variable @@error.

If *format-string* is not supplied or is empty, the error number is used to locate an error message in the system tables. Adaptive Server Enterprise obtains messages 17000-19999 from the SYSMESSAGES table. In Sybase IQ, this table is an empty view, so errors in this range should provide a format string. Messages for error numbers of 20000 or greater are obtained from the SYS.SYSUSERMESSAGES table.

The *format-string* can be up to 255 bytes long. This is the same as in Adaptive Server Enterprise.

The extended values supported by the SQL Server or Adaptive Server Enterprise RAISERROR statement are not supported in Sybase IQ.

The format string can contain placeholders for the arguments in the optional argument list. These placeholders are of the form *%nn!*, where *nn* is an integer between 1 and 20.

Intermediate RAISERROR status and code information is lost after the procedure terminates. If at return time an error occurs along with the RAISERROR then the error information is returned and the RAISERROR information is lost. The application can query intermediate RAISERROR statuses by examining @@error global variable at different execution points.

Side effects

None.

Standards

- **SQL92** Transact-SQL extension.
- **Sybase** Supported by Adaptive Server Enterprise.

Permissions

Must be connected to the database.

See also

CONTINUE_AFTER_RAISERROR option [TSQL] on page 53

ON_TSQL_ERROR option [TSQL] on page 128

READ statement [DBISQL]

Description

Reads DBISQL statements from a file.

Syntax

READ *filename* [*parameters*]

Examples

Examples of the READ statement:

```
READ status.rpt '160'
READ birthday.sql [>= '1988-1-1'] [<= '1988-1-30']
```

Usage

The READ statement reads a sequence of DBISQL statements from the named file. This file can contain any valid DBISQL statement, including other READ statements, which can be nested to any depth. To find the command file, DBISQL first searches the current directory, then the directories specified in the environment variable SQLPATH, then the directories specified in the environment variable PATH. If the named file has no file extension, DBISQL also searches each directory for the same file name with the extension *SQL*.

Parameters can be listed after the name of the command file. These parameters correspond to the parameters named on the PARAMETERS statement at the beginning of the statement file (see PARAMETERS statement [DBISQL] on page 610). DBISQL then substitutes the corresponding parameter wherever the source file contains:

```
{ parameter-name }
```

where *parameter-name* is the name of the appropriate parameter.

The parameters passed to a command file can be identifiers, numbers, quoted identifiers, or strings. When quotes are used around a parameter, the quotes are put into the text during the substitution. Parameters that are not identifiers, numbers, or strings (contain spaces or tabs) must be enclosed in square brackets ([ ]). This allows for arbitrary textual substitution in the command file.

If not enough parameters are passed to the command file, DBISQL prompts for values for the missing parameters.

Encoding

The READ statement also supports an ENCODING clause, which lets you specify the encoding that is used to read the file. For more information, see the READ statement in the *Adaptive Server Anywhere SQL Reference*.

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions

None.

See also

DEFAULT_ISQL_ENCODING option [DBISQL] on page 69

PARAMETERS statement [DBISQL] on page 610

RELEASE SAVEPOINT statement

Description
Releases a savepoint within the current transaction.

Syntax
RELEASE SAVEPOINT [*savepoint-name*]

Usage
The *savepoint-name* is an identifier specified on a SAVEPOINT statement within the current transaction. If *savepoint-name* is omitted, the most recent savepoint is released.

For a description of savepoints, see Chapter 8, "Using Procedures and Batches" in the *Sybase IQ System Administration Guide*. Releasing a savepoint does not perform any type of COMMIT; it simply removes the savepoint from the list of currently active savepoints.

Side effects

None.

Standards
- **SQL92** Vendor extension.
- **Sybase** Not supported by Adaptive Server Enterprise. A similar feature is available in an Adaptive Server Enterprise-compatible manner using nested transactions.

Permissions
There must have been a corresponding SAVEPOINT within the current transaction.

See also
ROLLBACK TO SAVEPOINT statement on page 631

SAVEPOINT statement on page 632

REMOVE statement

Description
Removes a class, a package, or a JAR file from a database. Removed classes are no longer available for use as a variable type.

Any class, package, or JAR to be removed must be already installed.

Syntax
REMOVE JAVA *classes_to_remove*

Parameters
classes_to_remove:
{ CLASS *java_class_name* [, *java_class_name*]...
| PACKAGE *java_package_name* [, *java_package_name*]...
| JAR *jar_name* [, *jar_name*]... [RETAIN CLASSES] }

jar_name:
character_string_expression

Examples

The following statement removes a Java class named “Demo” from the current database:

```
REMOVE JAVA CLASS Demo
```

Usage

java_class_name The name of one or more Java classes to be removed. Those classes must be installed classes in the current database.

java_package_name The name of one or more Java packages to be removed. Those packages must be the name of packages in the current database.

jar_name A character string value of maximum length 255.

Each *jar_name* must be equal to the *jar_name* of a retained JAR in the current database. Equality of *jar_name* is determined by the character string comparison rules of the SQL system.

If JAR...RETAIN CLASSES is specified, the specified JARs are no longer retained in the database, and the retained classes have no associated JAR. If RETAIN CLASSES is specified, this is the only action of the REMOVE statement.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not supported by Adaptive Server Enterprise. A similar feature is available in an Adaptive Server Enterprise-compatible manner using nested transactions.

Permissions

Must have DBA authority or must own the object.

RESIGNAL statement

Description

Resignals an exception condition.

Syntax

RESIGNAL [*exception-name*]

Examples

The following fragment returns all exceptions except for “Column Not Found” to the application.

```
...
DECLARE COLUMN_NOT_FOUND EXCEPTION
    FOR SQLSTATE '52003';
...
EXCEPTION
WHEN COLUMN_NOT_FOUND THEN
SET message='Column not found' ;
WHEN OTHERS THEN
```

```
RESIGNAL ;
```

Usage

Within an exception handler, RESIGNAL lets you quit the compound statement with the exception still active, or to quit reporting another named exception. The exception is handled by another exception handler or returned to the application. Any actions by the exception handler before the RESIGNAL are undone.

Side effects

None.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** Not supported in Adaptive Server Enterprise. Error handling in Transact-SQL procedures is carried out using the RAISERROR statement.

Permissions

None

See also

BEGIN... END statement on page 422

SIGNAL statement on page 652

RESTORE statement

Description

Restores a Sybase IQ database backup from one or more archive devices.

Syntax

RESTORE DATABASE '*db_file*'
FROM '*archive_device*' [**FROM** '*archive_device*']...
...[**KEY** *key_spec*]
... [**RENAME** *dbspace-name* **TO** '*new-dbspace-path*']...
... [**CATALOG ONLY**]

Parameters

db_file:
relative or absolute path of the database to be restored. Can be the original location, or a new location for the Catalog Store file.

key_spec:
quoted string including mixed cases, numbers, letters, and special characters. It might be necessary to protect the key from interpretation or alteration by the command shell.

Examples

- The following UNIX example restores the asiqdemo database from tape devices */dev/rmt/0* and */dev/rmt/2* on a Sun Solaris platform. On Solaris, the letter *n* after the device name specifies "no rewind on close." To specify this feature with RESTORE, use the naming convention appropriate for your UNIX platform. (Windows does not support this feature.)

```
RESTORE DATABASE 'asiqdemo'
FROM '/dev/rmt/0n'
FROM '/dev/rmt/2n'
```

- The following example restores an encrypted database named *marvin* that was encrypted with the key *is!seCret*.

```
RESTORE DATABASE 'marvin'
FROM 'marvin_bkup_file1'
FROM 'marvin_bkup_file2'
FROM 'marvin_bkup_file3'
KEY 'is!seCret'
```

Usage

The RESTORE command requires exclusive access by the DBA to the database. This exclusive access is achieved by setting the -gd switch to DBA, which is the default when you start the server engine. Issue the RESTORE command before you start the database (you must be connected to the utility_db database). Once you finish specifying RESTORE commands for the type of backup, that database is ready to be used. The database is left in the state that existed at the end of the first implicit CHECKPOINT of the last backup you restored. You can now specify a START DATABASE to allow other users to access the restored database.

Note You can restore only databases created using Sybase IQ 12.4.3 and higher. If the database was created using a 12.x version prior to 12.4.3, you must upgrade it to 12.4.3 or greater before backup. For instructions, see the *Sybase IQ Installation and Configuration Guide*for the version to which you are upgrading.

When restoring to a raw device, make sure the device is large enough to hold the dbspace you are restoring. IQ RESTORE checks the raw device size and returns an error, if the raw device is not large enough to restore the dbspace. For more information, see "Restoring to a raw device" in Chapter 14, "Data Backup, Recovery, and Archiving" in the *Sybase IQ System Administration Guide*.

BACKUP allows you to specify full or incremental backups. You can choose two kinds of incremental backups. INCREMENTAL backs up only those blocks that have changed and committed since the last backup of any type (incremental or full). INCREMENTAL SINCE FULL backs up all the blocks that have changed since the last full backup. If a RESTORE of a full backup is followed by one or more incremental backups (of either type), no modifications to the database are allowed between successive RESTORE commands. This rule prevents a RESTORE from incremental backups on a database in need of crash recovery, or one that has been modified. You can still overwrite such a database with a RESTORE from a full backup.

Before starting a full restore, you must delete two files: the Catalog Store file (default name *dbname.db)* and the transaction log file (default name *dbname.log)*.

If you restore an incremental backup, RESTORE ensures that backup media sets are accessed in the proper order. This order restores the last full backup tape set first, then the first incremental backup tape set, then the next most recent set, and so forth, until the most recent incremental backup tape set. If the DBA produced an INCREMENTAL SINCE FULL backup, only the full backup tape set and the most recent INCREMENTAL SINCE FULL backup tape set is required; however, if there is an INCREMENTAL made since the INCREMENTAL SINCE FULL, it also must be applied.

Sybase IQ ensures that the restoration order is appropriate, or it displays an error. Any other errors that occur during the restore results in the database being marked corrupt and unusable. To clean up a corrupt database, do a RESTORE from a full backup, followed by any additional incremental backups. Since the corruption probably happened with one of those backups, you might need to ignore a later backup set and use an earlier set.

FROM Specifies the name of the *archive_device* from which you are restoring, delimited with single quotation marks. If you are using multiple archive devices, specify them using separate FROM clauses. A comma-separated list is not allowed. Archive devices must be distinct. The number of FROM clauses determines the amount of parallelism Sybase IQ attempts with regard to input devices.

The backup/restore API DLL implementation lets you specify arguments to pass to the DLL when opening an archive device. For third-party implementations, the *archive_device* string has the following format:

```
'DLLidentifier::vendor_specific_information'
```

A specific example is:

```
'spsc::workorder=12;volname=ASD002'
```

The *archive_device* string length can be up to 1023 bytes. The *DLLidentifier* portion must be 1 to 30 bytes in length and can contain only alphanumeric and underscore characters. The *vendor_specific_information* portion of the string is passed to the third-party implementation without checking its contents.

Note Only certain third-party products are certified with Sybase IQ using this syntax. See the *Sybase IQ Release Bulletin* for additional usage instructions or restrictions. Before using any third-party product to back up your Sybase IQ database, make sure it is certified. See the *Sybase IQ Release Bulletin*, or see the Sybase Certification Reports for the Sybase IQ product in Technical Documents at http://www.sybase.com/support/techdocs/.

For the Sybase implementation of the backup/restore API, you need not specify information other than the tape device name or file name. However, if you use disk devices, you must specify the same number of archive devices on the RESTORE as given on the backup; otherwise, you may have a different number of restoration devices than the number used to perform the backup. A specific example of an archive device for the Sybase API DLL that specifies a nonrewinding tape device for a UNIX system is:

```
'/dev/rmt/0n'
```

RENAME Lets you restore one or more Sybase IQ database files to a new location. Specify each *dbspace-name* you are moving as it appears in the SYSFILE table. Specify *new-dbspace-path* as the new raw partition, or the new full or relative path name, for that dbspace.

If relative paths were used to create the database files, the files are restored by default relative to the Catalog Store file (the SYSTEM dbspace), and a rename clause is not required. If absolute paths were used to create the database files and a rename clause is not specified for a file, it is restored to its original location.

Relative path names in the RENAME clause work as they do when you create a database or dbspace: the main IQ Store dbspace, Temporary Store dbspaces, and Message Log are restored relative to the location of *db_file* (the Catalog Store); user-created IQ Store dbspaces are restored relative to the directory that holds the main IQ dbspace.

Do not use the RENAME clause to move the SYSTEM dbspace, which holds the Catalog Store. To move the Catalog Store, and any files created relative to it and not specified in a RENAME clause, specify a new location in the *db_file* parameter.

CATALOG ONLY Restores only the backup header record from the archive media.

Other RESTORE issues:

- RESTORE to disk does not support raw devices as archival devices.
- Sybase IQ does not rewind tapes before using them; on rewinding tape devices, it does rewind tapes after using them. You must position each tape to the start of the Sybase IQ data before starting the RESTORE.
- During BACKUP and RESTORE operations, if Sybase IQ cannot open the archive device (for example, when it needs the media loaded) and the ATTENDED option is ON, it waits for ten seconds for you to put the next tape in the drive, and then tries again. It continues these attempts indefinitely until either it is successful or the operation is terminated with Ctrl+C.
- If you press Ctrl+C, RESTORE fails and returns the database to its state before the restoration began.
- If disk striping is used, the striped disks are treated as a single device.
- The file_name column in the SYSFILE system table for the SYSTEM dbspace is not updated during a restore. For the SYSTEM dbspace, the file_name column always reflects the name when the database was created. The filename of the SYSTEM dbspace is the name of the database file.

The maximum size for a complete RESTORE command, including all clauses, is 32KB.

Side effects

None.

Standards

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

Permissions

Must have DBA authority.

See also

BACKUP statement on page 416

RESUME statement

Description

Resumes a procedure after a query.

Syntax

Syntax 1

RESUME *cursor-name*

Syntax 2

RESUME [**ALL**]

Parameters

cursor-name:
identifier

cursor-name:
identifier or host-variable

Examples

- Embedded SQL examples:

```
1. EXEC SQL RESUME cur_employee;
2. EXEC SQL RESUME :cursor_var;
```

- DBISQL examples:

```
CALL sample_proc() ;
RESUME ALL;
```

Usage

The RESUME statement resumes execution of a procedure that returns result sets. The procedure executes until the next result set (SELECT statement with no INTO clause) is encountered. If the procedure completes and no result set is found, the SQLSTATE_PROCEDURE_COMPLETE warning is set. This warning is also set when you RESUME a cursor for a SELECT statement.

Note The RESUME statement is supported in dbisqlc, but is invalid in dbisql (Interactive SQL Java) or when connected to the database using the iAnywhere JDBC driver.

The DBISQL RESUME statement (Format 2) resumes the current procedure. If ALL is not specified, executing RESUME displays the next result set or, if no more result sets are returned, completes the procedure.

The DBISQL RESUME ALL statement cycles through all result sets in a procedure, without displaying them, and completes the procedure. This is useful mainly in testing procedures.

Side effects

None.

Standards

- **SQL92** Vendor extension.

- **Sybase** Not supported by Adaptive Server Enterprise.

Permissions — The cursor must have been previously opened.

See also — DECLARE CURSOR statement [ESQL] [SP] on page 516

RETURN statement

Description — Exits a function or procedure unconditionally, optionally providing a return value. Statements following RETURN are not executed.

Syntax — **RETURN** [(*expression*)]

Examples

- Returns the product of three numbers:

```
CREATE FUNCTION product ( a numeric,
                          b numeric ,
                          c numeric)
RETURNS numeric
BEGIN
   RETURN ( a * b * c ) ;
END
```

- Calculates the product of three numbers:

```
SELECT product (2, 3, 4)
```

```
product (2,3,4)
24
```

- Uses the RETURN statement to avoid executing a complex query if it is meaningless:

```
CREATE PROCEDURE customer_products
( in customer_id integer DEFAULT NULL)
RESULT ( id integer, quantity_ordered integer )
BEGIN
   IF customer_id NOT IN (SELECT id FROM customer)
   OR customer_id IS NULL THEN
      RETURN
   ELSE
      SELECT product.id,sum(
         sales_order_items.quantity )
      FROM  product,
            sales_order_items,
            sales_order
      WHERE sales_order.cust_id=customer_id
```

```
            AND sales_order.id=sales_order_items.id
            AND sales_order_items.prod_id=product.id
            GROUP BY product.id
        END IF
    END
```

Usage

If *expression* is supplied, the value of *expression* is returned as the value of the function or procedure.

Within a function, the expression should be of the same data type as the function's RETURNS data type.

RETURN is used in procedures for Transact-SQL-compatibility, and is used to return an integer error code.

Side effects

None.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** Transact-SQL procedures use the return statement to return an integer error code.

Permissions

None.

See also

BEGIN... END statement on page 422

CREATE PROCEDURE statement on page 485

REVOKE statement

Description

Removes permissions for specified users.

Syntax

Syntax 1

REVOKE
{ **CONNECT** | **DBA** | **INTEGRATED LOGIN** | **GROUP**
| **MEMBERSHIP IN GROUP** *userid* [, ...] | **RESOURCE** }
... **FROM** *userid* [, ...]

Syntax 2

REVOKE
{ **ALL** [**PRIVILEGES**] | **ALTER** | **DELETE** | **INSERT**
| **REFERENCE** | **SELECT** [(*column-name* [, ...])] | **UPDATE** [(*column-name*,...)] }
... **ON** [*owner*.]*table-name* **FROM** *userid* [, ...]

Syntax 3

REVOKE EXECUTE ON [*owner*.]*procedure-name* **FROM** *userid* [, ...]

Examples

- Prevents user “dave” from inserting into the employee table:

```
REVOKE INSERT ON employee FROM dave ;
```

- Revokes resource permission from user “Jim”:

```
REVOKE RESOURCE FROM Jim ;
```

- Prevents user “dave” from updating the employee table:

```
REVOKE UPDATE ON employee FROM dave ;
```

- Revokes integrated login mapping from the user profile name “Administrator”:

```
REVOKE INTEGRATED LOGIN FROM Administrator ;
```

- Disallows the finance group from executing the procedure sp_customer_list:

```
REVOKE EXECUTE ON sp_customer_list
FROM finance ;
```

- Drops user ID “FranW” from the database:

```
REVOKE CONNECT FROM FranW ;
```

Usage

The REVOKE statement is used to remove permissions that were given using the GRANT statement. Syntax 1 is used to revoke special user permissions and Syntax 2 is used to revoke table permissions. Syntax 3 is used to revoke permission to execute a procedure. REVOKE CONNECT is used to remove a user ID from a database.

Note If Login Management is enabled for the database, you must use system procedures, not GRANT and REVOKE, to add and remove user IDs.

REVOKE GROUP automatically revokes membership from all members of the group.

By default, you can only remove users with REVOKE CONNECT on a multiplex write server. To enable REVOKE CONNECT on query servers, you must set the database option MPX_LOCAL_SPEC_PRIV to change the default. For details, see "MPX_LOCAL_SPEC_PRIV option" on page 123.

Note You cannot revoke a user's connect privileges if that user owns database objects, such as tables. Attempting to do so with a REVOKE statement or sp_dropuser procedure returns an error such as "Cannot drop a user that owns tables in runtime system."

Side effects

Automatic commit.

Standards

- **SQL92** Syntax 1 is a vendor extension. Syntax 2 is an entry-level feature. Syntax 3 is a Persistent Stored Module feature.
- **Sybase** Syntax 2 and 3 are supported by Adaptive Server Enterprise. Syntax 1 is not supported by Adaptive Server Enterprise. User management and security models are different for Sybase IQ and Adaptive Server Enterprise.

Permissions

Must be the grantor of the permissions that are being revoked, or must have DBA authority.

If revoking CONNECT permissions or revoking table permissions from another user, the other user must not be connected to the database.

See also

GRANT statement on page 559

ROLLBACK statement

Description

Undoes any changes made since the last COMMIT or ROLLBACK.

Syntax

ROLLBACK [**WORK**]

Usage

ROLLBACK ends a logical unit of work (transaction) and undoes all changes made to the database during this transaction. A transaction is the database work done between COMMIT or ROLLBACK statements on one database connection.

Side effects

Closes all cursors not opened WITH HOLD.

Releases locks held by the transaction issuing the ROLLBACK.

Standards
- **SQL92** Entry-level feature.
- **Sybase** Supported by Adaptive Server Enterprise.

Permissions: Must be connected to the database.

See also: COMMIT statement on page 436

ROLLBACK TO SAVEPOINT statement on page 631

ROLLBACK TO SAVEPOINT statement

Description: Cancels any changes made since a SAVEPOINT.

Syntax: **ROLLBACK TO SAVEPOINT** [*savepoint-name*]

Usage: Changes made prior to the SAVEPOINT are not undone; they are still pending. For a description of savepoints, see Chapter 8, "Using Procedures and Batches" in the *Sybase IQ System Administration Guide*.

The *savepoint-name* is an identifier that was specified on a SAVEPOINT statement within the current transaction. If *savepoint-name* is omitted, the most recent savepoint is used. Any savepoints since the named savepoint are automatically released.

Side effects

None.

Standards
- **SQL92** Vendor extension.
- **Sybase** Savepoints are not supported by Adaptive Server Enterprise. To implement similar features in an Adaptive Server Enterprise-compatible manner, you can use nested transactions.

Permissions: There must have been a corresponding SAVEPOINT within the current transaction.

See also: RELEASE SAVEPOINT statement on page 619

ROLLBACK statement on page 630

SAVEPOINT statement on page 632

SAVEPOINT statement

Description
Establishes a savepoint within the current transaction.

Syntax
SAVEPOINT [*savepoint-name*]

Usage
The *savepoint-name* is an identifier that can be used in a RELEASE SAVEPOINT or ROLLBACK TO SAVEPOINT statement. All savepoints are automatically released when a transaction ends. See Chapter 8, "Using Procedures and Batches" in the *Sybase IQ System Administration Guide*.

Savepoints that are established while a trigger is executing or while an atomic compound statement is executing are automatically released when the atomic operation ends.

Side effects

None.

Standards
- **SQL92** Vendor extension.
- **Sybase** Not supported in Adaptive Server Enterprise. To implement similar features in an Adaptive Server Enterprise-compatible manner, use nested transactions.

Permissions
None

See also
RELEASE SAVEPOINT statement on page 619

ROLLBACK TO SAVEPOINT statement on page 631

SELECT statement

Description
Retrieves information from the database.

Syntax
SELECT [**ALL** | **DISTINCT**] [**FIRST** | **TOP** *number-of-rows*] *select-list*
... [**INTO** { *host-variable-list* | *variable-list* | *table-name* }]
... [**FROM** *table-list*]
... [**WHERE** *search-condition*]
... [**GROUP BY** [*expression* [,...]
| **ROLLUP** (*expression* [,...])
| **CUBE** (*expression* [,...])]]
... [**HAVING** *search-condition*]
... [**ORDER BY** { *expression* | *integer* } [**ASC** | **DESC**] [, ...]]

Parameters

select-list:
{ *column-name*
| *expression* [[AS] *alias-name*]
| * }

Examples

- Lists all the tables and views in the system catalog:

```
SELECT tname
FROM SYS.SYSCATALOG
WHERE tname LIKE 'SYS%' ;
```

- Lists all customers and the total value of their orders:

```
SELECT company_name,
    CAST( sum(sales_order_items.quantity *
    product.unit_price) AS INTEGER) VALUE
FROM customer
    LEFT OUTER JOIN sales_order
    LEFT OUTER JOIN sales_order_items
    LEFT OUTER JOIN product
GROUP BY company_name
ORDER BY VALUE DESC
```

- Lists the number of employees:

```
SELECT count(*)
FROM Employee ;
```

- Shows an Embedded SQL SELECT statement:

```
SELECT count(*) INTO :size FROM employee
```

- Lists the total sales by year, model, and color:

```
SELECT year, model, color, sum(sales) FROM sales_tab
GROUP BY ROLLUP (year, model, color);
```

- Selects all items with a certain discount into a temporary table:

```
SELECT * INTO #TableTemp FROM lineitem WHERE l_discount
< 0.5
```

Usage

You can use a SELECT statement in DBISQL to browse data in the database or to export data from the database to an external file.

You can also use a SELECT statement in procedures or in Embedded SQL. The SELECT statement with an INTO clause is used for retrieving results from the database when the SELECT statement returns only one row. (Tables created with SELECT INTO do not inherit IDENTITY/AUTOINCREMENT tables.) For multiple-row queries, you must use cursors.When you select more than one column and do not use *#table*, SELECT INTO creates a permanent base table. SELECT INTO *#table* always creates a temporary table regardless of the number of columns. SELECT INTO table with a single column selects into a host variable.

Tables with the same name but different owners require aliases. A query like the following returns incorrect results:

```
SELECT * FROM user1.t1
WHERE NOT EXISTS
(SELECT *
FROM user2.t1
WHERE user2.t1.col1 = user1.t.col1);
```

For correct results, use an alias for each table, as follows:

```
SELECT * FROM user1.t1 U1
WHERE NOT EXISTS
(SELECT *
FROM user2.t1 U2
WHERE U2.col1 = U1.col1);
```

The INTO clause with a *variable-list* is used in procedures only.

A SELECT statement can also return a result set from a procedure. See "Creating and selecting from temporary tables" in the *Sybase IQ System Administration Guide* for a restriction that affects selecting from temporary tables within stored procedures.

The various parts of the SELECT statement are described below:

ALL or DISTINCT If neither is specified, all rows that satisfy the clauses of the SELECT statement are retrieved. If DISTINCT is specified, duplicate output rows are eliminated. This is called the projection of the result of the statement. In many cases, statements take significantly longer to execute when DISTINCT is specified, so reserve the use of DISTINCT for cases where it is necessary.

If DISTINCT is used, the statement cannot contain an aggregate function with a DISTINCT parameter.

FIRST or TOP number-of-rows Specifies the number of rows returned from a query. FIRST returns the first row selected from the query. TOP returns the specified number of rows from the query, where *number-of-rows* is in the range 1 – 32767, and can be an integer constant or integer variable.

FIRST and TOP are used primarily with the ORDER BY clause. If you use these keywords without an ORDER BY clause, the result might vary from run to run of the same query, as the optimizer might choose a different query plan.

FIRST and TOP are permitted only in the top-level SELECT of a query, so they cannot be used in derived tables or view definitions. Using FIRST or TOP in a view definition might result in the keyword being ignored when a query is run on the view.

Using FIRST is the same as setting the ROW_COUNT database option to 1. Using TOP is the same as setting the ROW_COUNT option to the same number of rows, except that the maximum number of rows returned for TOP is 32767. ROW_COUNT does not have an upper limit for the number of rows returned. If both TOP and ROW_COUNT are set, then the value of TOP takes precedence.

If you need the query to return more than 32K rows, use ROW_COUNT. For more information on the ROW_COUNT database option, see "ROW_COUNT option" on page 144.

select-list The *select-list* is a list of expressions, separated by commas, specifying what is retrieved from the database. If an asterisk (*) is specified, all columns of all tables in the FROM clause (table-name all columns of the named table) are selected. Aggregate functions and analytical functions are allowed in the *select-list* (see Chapter 5, "SQL Functions").

Note In Sybase IQ, scalar subqueries (nested selects) are allowed in the select list of the top level SELECT, as in Adaptive Server Anywhere and Adaptive Server Enterprise. Subqueries cannot be used inside a conditional value expression (for example, in a CASE statement).

In Sybase IQ, subqueries can also be used in a WHERE or HAVING clause predicate (one of the supported predicate types). However, inside the WHERE or HAVING clause, subqueries cannot be used inside a value expression or inside a CONTAINS or LIKE predicate. Subqueries are not allowed in the ON clause of outer joins or in the GROUP BY clause.

For more details on the use of subqueries, see "Subqueries in expressions" on page 181 and "Subqueries in search conditions" on page 191.

---

alias-names can be used throughout the query to represent the aliased expression. Alias names are also displayed by DBISQL at the top of each column of output from the SELECT statement. If the optional *alias-name* is not specified after an expression, DBISQL displays the expression. You cannot use the same name or expression for a column alias as the column name; Sybase IQ prevents this usage because it would be a recursive reference.

INTO host-variable-list This clause is used in Embedded SQL only. It specifies where the results of the SELECT statement goes. There must be one *host-variable* item for each item in the *select-list*. Select list items are put into the host variables in order. An indicator host variable is also allowed with each *host-variable* so the program can tell if the select list item was NULL.

INTO variable-list This clause is used in procedures only. It specifies where the results of the SELECT statement go. There must be one variable for each item in the select list. Select list items are put into the variables in order.

INTO table-name This clause is used to create a table and fill it with data.

If the table name starts with #, the table is created as a temporary table. Otherwise, the table is created as a permanent base table. For permanent tables to be created, the query must satisfy the following conditions:

- The *select-list* contains more than one item, and the INTO target is a single *table-name* identifier, or
- The select-list contains a * and the INTO target is specified as *owner.table*.

To create a permanent table with one column, the table name must be specified as *owner.table*. Omit the owner specification for a temporary table.

This statement causes a COMMIT before execution as a side effect of creating the table. RESOURCE authority is required to execute this statement. No permissions are granted on the new table: the statement is a short form for CREATE TABLE followed by INSERT... SELECT.

Tables created using this statement do not have a primary key defined. You can add a primary key using ALTER TABLE. A primary key should be added before applying any UPDATEs or DELETEs to the table; otherwise, these operations result in all column values being logged in the transaction log for the affected rows.

Use of this clause is restricted to valid Adaptive Server Anywhere queries. Sybase IQ extensions are not supported.

FROM table-list Rows are retrieved from the tables and views specified in the *table-list*. Joins can be specified using join operators. For more information, see FROM clause on page 553. A SELECT statement with no FROM clause can be used to display the values of expressions not derived from tables. For example:

```
SELECT @@version
```

displays the value of the global variable @@version. This is equivalent to:

```
SELECT @@version
FROM DUMMY
```

Note If you omit the FROM clause, or if all tables in the query are in the SYSTEM dbspace, the query is processed by Adaptive Server Anywhere instead of Sybase IQ and might behave differently, especially with respect to syntactic and semantic restrictions and the effects of option settings. See the Adaptive Server Anywhere documentation for rules that might apply to processing.

If you have a query that does not require a FROM clause, you can force the query to be processed by Sybase IQ by adding the clause "FROM iq_dummy," where iq_dummy is a one-row, one-column table that you create in your database.

WHERE search-condition Specifies which rows are selected from the tables named in the FROM clause. It is also used to do joins between multiple tables. This is accomplished by putting a condition in the WHERE clause that relates a column or group of columns from one table with a column or group of columns from another table. Both tables must be listed in the FROM clause.

The use of the same CASE statement is not allowed in both the SELECT and the WHERE clause of a grouped query. See "Search conditions" on page 189 for a full description.

GROUP BY You can group by columns or alias names or functions. GROUP BY expressions must also appear in the select list. The result of the query contains one row for each distinct set of values in the named columns, aliases, or functions. The resulting rows are often referred to as *groups* since there is one row in the result for each group of rows from the table list. For the sake of GROUP BY, all NULL values are treated as identical. Aggregate functions can then be applied to these groups to get meaningful results.

GROUP BY must contain more than a single constant. You do not need to add constants to the GROUP BY clause to select the constants in grouped queries. If the GROUP BY expression contains only a single constan, an error is returned and the query is rejected.

When GROUP BY is used, the select list, HAVING clause, and ORDER BY clause cannot reference any identifiers except those named in the GROUP BY clause. The following exception applies: The *select-list* and HAVING clause may contain aggregate functions.

ROLLUP operator The ROLLUP operator in the GROUP BY clause lets you analyze subtotals using different levels of detail. It creates subtotals that roll up from a detailed level to a grand total.

The ROLLUP operator requires an ordered list of grouping expressions to be supplied as arguments. ROLLUP first calculates the standard aggregate values specified in the GROUP BY. Then ROLLUP moves from right to left through the list of grouping columns and creates progressively higher-level subtotals. A grand total is created at the end. If *n* is the number of grouping columns, ROLLUP creates *n+1* levels of subtotals.

Restrictions on the ROLLUP operator are:

- The ROLLUP operator supports all of the aggregate functions available to the GROUP BY clause, but ROLLUP does not currently support COUNT DISTINCT and SUM DISTINCT.
- ROLLUP can be used only in the SELECT statement; you cannot use ROLLUP in a SELECT subquery.
- A multiple grouping specification that combines ROLLUP, CUBE, and GROUP BY columns in the same GROUP BY clause is not currently supported.
- Constant expressions as GROUP BY keys are not supported.

GROUPING is used with the ROLLUP operator to distinguish between stored NULL values and NULL values in query results created by ROLLUP.

ROLLUP syntax:

SELECT ... [**GROUPING** (*column-name*) ...] ...
GROUP BY [*expression* [,...]
| **ROLLUP** (*expression* [,...])]

See “Expressions” on page 179 in Chapter 3, “SQL Language Elements” for the format of an operator expression.

GROUPING takes a column name as a parameter and returns a Boolean value as listed in Table 6-12.

Table 6-12: Values returned by GROUPING with the ROLLUP operator

If the value of the result is	GROUPING returns
NULL created by a ROLLUP operation	1 (TRUE)
NULL indicating the row is a subtotal	1 (TRUE)
not created by a ROLLUP operation	0 (FALSE)
a stored NULL	0 (FALSE)

For ROLLUP examples, see Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*.

CUBE operator The CUBE operator in the GROUP BY clause analyzes data by forming the data into groups in more than one dimension. CUBE requires an ordered list of grouping expressions (dimensions) as arguments and enables the SELECT statement to calculate subtotals for all possible combinations of the group of dimensions.

Restrictions on the CUBE operator are:

- The CUBE operator supports all of the aggregate functions available to the GROUP BY clause, but CUBE does not currently support COUNT DISTINCT or SUM DISTINCT.
- CUBE does not currently support the inverse distribution analytical functions, PERCENTILE_CONT and PERCENTILE_DISC.
- CUBE can be used only in the SELECT statement; you cannot use CUBE in a SELECT subquery.
- A multiple GROUPING specification that combines ROLLUP, CUBE, and GROUP BY columns in the same GROUP BY clause is not currently supported.
- Constant expressions as GROUP BY keys are not supported.

GROUPING is used with the CUBE operator to distinguish between stored NULL values and NULL values in query results created by CUBE.

CUBE syntax:

SELECT ... [**GROUPING** (*column-name*) ...] ...
GROUP BY [*expression* [,...]
| **CUBE** (*expression* [,...])]

GROUPING takes a column name as a parameter and returns a Boolean value as listed in Table 6-13.

Table 6-13: Values returned by GROUPING with the CUBE operator

If the value of the result is	GROUPING returns
NULL created by a CUBE operation	1 (TRUE)
NULL indicating the row is a subtotal	1 (TRUE)
not created by a CUBE operation	0 (FALSE)
a stored NULL	0 (FALSE)

When generating a query plan, the IQ optimizer estimates the total number of groups generated by the GROUP BY CUBE hash operation. The MAX_CUBE_RESULTS database option sets an upper boundary for the number of estimated rows the optimizer considers for a hash algorithm that can be run. If the actual number of rows exceeds the MAX_CUBE_RESULT option value, the optimizer stops processing the query and returns the error message "Estimate number: nnn exceed the DEFAULT_MAX_CUBE_RESULT of GROUP BY CUBE or ROLLUP", where nnn is the number estimated by the IQ optimizer. See "MAX_CUBE_RESULT option" in Chapter 2, "Database Options" for information on setting the MAX_CUBE_RESULT option.

For CUBE examples, see Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*.

HAVING search-condition Based on the group values and not on the individual row values. The HAVING clause can be used only if either the statement has a GROUP BY clause or if the select list consists solely of aggregate functions. Any column names referenced in the HAVING clause must either be in the GROUP BY clause or be used as a parameter to an aggregate function in the HAVING clause.

ORDER BY Orders the results of a query. Each item in the ORDER BY list can be labeled as ASC for ascending order or DESC for descending order. Ascending is assumed if neither is specified. If the expression is an integer n, then the query results are sorted by the nth item in the select list.

In Embedded SQL, the SELECT statement is used for retrieving results from the database and placing the values into host variables with the INTO clause. The SELECT statement must return only one row. For multiple row queries, you must use cursors.

You cannot include a Java class in the SELECT list, but you can, for example, create a function or variable that acts as a wrapper for the Java class and then select it.

Side effects

None.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Adaptive Server Enterprise, with some differences in syntax.

Permissions

Must have SELECT permission on the named tables and views.

See also

CREATE VIEW statement on page 512

DECLARE CURSOR statement [ESQL] [SP] on page 516

"Expressions" on page 179

FETCH statement [ESQL] [SP] on page 547

FROM clause on page 553

OPEN statement [ESQL] [SP] on page 603

"Search conditions" on page 189

UNION operation on page 659

Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

"Access fields and methods of the Java object" in the *Adaptive Server Anywhere Programming Guide* chapter, "Introduction to Java in the Database"

SET statement

Description

Assigns a value to a SQL variable.

Syntax

SET *identifier = expression*

Examples

- The following code fragment can be used to insert a large text value into the database:

```
EXEC SQL BEGIN DECLARE SECTION;
char buffer[5001];
EXEC SQL END DECLARE SECTION;


EXEC SQL CREATE VARIABLE hold_text VARCHAR;
EXEC SQL SET hold_text = '';
for(;;) {
    /* read some data into buffer ... */
    size = fread( buffer, 1, 5000, fp );
    if( size <= 0 ) break;
```

```
    /* buffer must be null-terminated */
    buffer[size] = '\0';
    /* add data to blob using concatenation */
    EXEC SQL SET hold_text = hold_text || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES ( 1, hold_text );
EXEC SQL DROP VARIABLE hold_text;
```

- The following code fragment can be used to insert a large binary value into the database:

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_BINARY( 5000 ) buffer;
EXEC SQL END DECLARE SECTION;
EXEC SQL CREATE VARIABLE hold_blob LONG BINARY;
EXEC SQL SET hold_blob = '';
for(;;) {
    /* read some data into buffer ... */
    size = fread( &(buffer.array), 1, 5000, fp );
    if( size <= 0 ) break;
    buffer.len = size;


    /* add data to blob using concatenation
       Note that concatenation works for
       binary data too! */
    EXEC SQL SET hold_blob = hold_blob || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES ( 1, hold_blob );
EXEC SQL DROP VARIABLE hold_blob;
```

Usage

The SET statement assigns a new value to a variable that was previously created using the CREATE VARIABLE statement.

You can use a variable in a SQL statement anywhere a column name is allowed. If there is no column name that matches the identifier, the database server checks to see if there is a variable that matches, and uses its value.

Variables are local to the current connection, and disappear when you disconnect from the database or when you use DROP VARIABLE. They are not affected by COMMIT or ROLLBACK statements.

Variables are necessary for creating large text or binary objects for INSERT or UPDATE statements from Embedded SQL programs because Embedded SQL host variables are limited to 32,767 bytes.

Side effects

None.

Standards

- **SQL92** Persistent Stored Module feature.
- **Sybase** Not supported. In Adaptive Server Enterprise, variables are assigned using the SELECT statement with no table, a Transact-SQL syntax that is also supported by Sybase IQ. The SET statement is used to set database options in Adaptive Server Enterprise.

Permissions

None.

See also

CREATE VARIABLE statement on page 511

DROP VARIABLE statement on page 540

"Expressions" on page 179

SET statement [T-SQL]

Description

Sets database options in an Adaptive Server Enterprise-compatible manner.

Syntax

SET *option-name option-value*

Usage

Table 6-14 lists available options.

Table 6-14: Transact-SQL SET options

Option name	Option value
ANSINULL	ON | OFF
ANSI_PERMISSIONS	ON | OFF
CLOSE_ON_ENDTRANS	ON | OFF
QUOTED_IDENTIFIER	ON | OFF
ROWCOUNT	*integer*
STRING_RTRUNCATION	ON | OFF
TRANSACTION ISOLATION LEVEL	0 | 1 | 2 | 3

Database options in Sybase IQ are set using the SET OPTION statement. However, Sybase IQ also provides support for the Adaptive Server Enterprise SET statement for a set of options particularly useful for compatibility.

You can set the following options using the Transact-SQL SET statement in Sybase IQ, as well as in Adaptive Server Enterprise:

- **SET ANSINULL { ON | OFF }** The default behavior for comparing values to NULL in Sybase IQ and Adaptive Server Enterprise is different. Setting ANSINULL to OFF provides Transact-SQL compatible comparisons with NULL

- **SET ANSI_PERMISSIONS { ON | OFF }** The default behavior in Sybase IQ and Adaptive Server Enterprise regarding permissions required to carry out a DELETE containing a column reference is different. Setting ANSI_PERMISSIONS to OFF provides Transact-SQL-compatible permissions on DELETE.
- **SET CLOSE_ON_ENDTRANS { ON | OFF }** The default behavior in Sybase IQ and Adaptive Server Enterprise for closing cursors at the end of a transaction is different. Setting CLOSE_ON_ENDTRANS to OFF provides Transact-SQL-compatible behavior.
- **SET QUOTED_IDENTIFIER { ON | OFF }** Controls whether strings enclosed in double quotes are interpreted as identifiers (ON) or as literal strings (OFF).
- **SET ROWCOUNT** *integer* The Transact-SQL ROWCOUNT option limits to the specified integer the number of rows fetched for any cursor. This includes rows fetched by repositioning the cursor. Any fetches beyond this maximum return a warning. The option setting is considered when returning the estimate of the number of rows for a cursor on an OPEN request.

  Note The ROWCOUNT option has no effect on UPDATE and DELETE operations in Sybase IQ. Also note that Sybase IQ does not support the *@@rowcount* global variable.

In Sybase IQ, if ROWCOUNT is greater than the number of rows that DBISQL can display, DBISQL may do some extra fetches to reposition the cursor. Thus, the number of rows actually displayed may be less than the number requested. Also, if any rows are refetched due to truncation warnings, the count might be inaccurate.

A value of zero resets the option to get all rows.

- **SET STRING_RTRUNCATION { ON | OFF }** The default behavior in Sybase IQ and Adaptive Server Enterprise when nonspace characters are truncated on assigning SQL string data is different. Setting STRING_RTRUNCATION to ON provides Transact-SQL-compatible string comparisons, including hexadecimal string (binary data type) comparisons.

- **SET TRANSACTION ISOLATION LEVEL { 0 | 1 | 2 | 3 }** Sets the locking isolation level for the current connection, as described in Chapter 10, "Transactions and Versioning" in the *Sybase IQ System Administration Guide*. For Adaptive Server Enterprise, only 1 and 3 are valid options. For Sybase IQ, only 3 is a valid option.

In addition, the following SET statement is allowed by Sybase IQ for compatibility, but has no effect:

- **SET PREFETCH {ON | OFF}**

Side effects

None.

Standards

- **SQL92** Transact-SQL extension.
- **Sybase** Sybase IQ supports a subset of the Adaptive Server Enterprise database options.

Permissions

None

See also

SET OPTION statement on page 647

SET CONNECTION statement [DBISQL] [ESQL]

Description

Changes the active database connection.

Syntax

SET CONNECTION [*connection-name*]

Parameters

connection-name:
identifier, string or host-variable

Examples

- In Embedded SQL:

```
EXEC SQL SET CONNECTION :conn_name;
```

- From DBISQL, sets the current connection to the connection named "conn1":

```
SET CONNECTION conn1 ;
```

Usage

The current connection state is saved and is resumed when it again becomes the active connection. If *connection-name* is omitted and there is a connection that was not named, that connection becomes the active connection.

Note When cursors are opened in Embedded SQL, they are associated with the current connection. When the connection is changed, the cursor names are not accessible. The cursors remain active and in position and become accessible when the associated connection becomes active again.

Side effects

None.

Standards

- **SQL92** DBISQL use is a vendor extension. Embedded SQL is a full-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions

None.

See also

CONNECT statement [ESQL] [DBISQL] on page 439

DISCONNECT statement [DBISQL] on page 532

SET DESCRIPTOR statement [ESQL]

Description

Describes the variables in a SQL descriptor area, and places data into the descriptor area.

Syntax

SET DESCRIPTOR *descriptor-name*
... { **COUNT** = { *integer* | *hostvar* }
| **VALUE** *n assignment* [, ...] }

Parameters

assignment:
{ { TYPE | SCALE | PRECISION | LENGTH | INDICATOR }
= { *integer* | *hostvar* }
| DATA = *hostvar* }

Examples

For an example, see ALLOCATE DESCRIPTOR statement [ESQL] on page 394.

Usage

SET...COUNT sets the number of described variables within the descriptor area. The value for count cannot exceed the number of variables specified when the descriptor area was allocated.

The value *n* specifies the variable in the descriptor area upon which the assignments are performed.

Type checking is performed when doing SET...DATA to ensure that the variable in the descriptor area has the same type as the host variable.

If an error occurs, the code is returned in the SQLCA.

Side effects

None.

Standards

- **SQL92** Intermediate-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions

None

See also

DEALLOCATE DESCRIPTOR statement [ESQL] on page 514

SET OPTION statement

Description

Changes database options.

Syntax

SET [**EXISTING**] [**TEMPORARY**] **OPTION**
... [*userid*. | **PUBLIC**.]*option-name* = [*option-value*]

Parameters

userid:
identifier, string, or host-variable

option-name:
identifier, string, or host-variable

option-value:
host-variable (indicator allowed), string, identifier, or number

Examples

- Sets the DATE_FORMAT option:

```
SET OPTION public.date_format = 'Mmm dd yyyy' ;
```

- Sets the WAIT_FOR_COMMIT option to on:

```
SET OPTION wait_for_commit = 'on' ;
```

- Embedded SQL examples:

```
1. EXEC SQL SET OPTION :user.:option_name = :value;
2. EXEC SQL SET TEMPORARY OPTION Date_format = 'mm/dd/
yyyy';
```

Usage

The SET OPTION statement is used to change options that affect the behavior of the database and its compatibility with Transact-SQL. Setting the value of an option can change the behavior for all users or an individual user, in either a temporary or permanent scope.

The classes of options are:

- General database options
- Transact-SQL compatibility database options

Specifying either a user ID or the PUBLIC user ID determines whether the option is set for an individual user, a user group represented by *userid*, or the PUBLIC user ID (the user group to which all users are a member). If no user group is specified, the option change is applied to the currently logged-on user ID that issued the SET OPTION statement.

For example, the following statement applies an option change to the user DBA, if DBA is the user issuing the SQL statement:

```
SET OPTION login_mode = mixed
```

However, the following statement applies the change to the PUBLIC user ID, a user group to which all users belong:

```
SET OPTION Public.login_mode = standard
```

Only users with DBA privileges have the authority to set an option for the PUBLIC user ID.

In Embedded SQL, only database options can be set temporarily.

Changing the value of an option for the PUBLIC user ID sets the value of the option for any user that has not set its own value. Option values cannot be set for an individual user ID unless there is already a PUBLIC user ID setting for that option.

Users cannot set the options of another user, unless they have DBA authority.

Users can use the SET OPTION statement to change the values for their own user IDs. Setting the value of an option for a user ID other then your own is permitted only if you have DBA authority.

If you use the EXISTING keyword, option values cannot be set for an individual user ID unless there is already a PUBLIC user ID setting for that option.

Adding the TEMPORARY keyword to the SET OPTION statement changes the duration that the change takes effect. Without the TEMPORARY keyword, an option change is permanent: it does not change until it is explicitly changed using SET OPTION.

When SET TEMPORARY OPTION is applied using an individual user ID, the new option value is in effect as long as that user is logged in to the database.

When SET TEMPORARY OPTION is used with the PUBLIC user ID, the change is in place for as long as the database is running. When the database is shut down, TEMPORARY options for the PUBLIC user ID revert back to their permanent value.

Temporarily setting an option for the PUBLIC user ID as opposed to setting the value of the option permanently offers a security advantage. For example, when the login_mode option is enabled, the database relies on the login security of the system on which it is running. Enabling the option temporarily means a database relying on the security of a Windows domain is not compromised if the database is shut down and copied to a local machine. In that case, the temporary enabling of login_mode reverts to its permanent value, which might be Standard, a mode in which integrated logins are not permitted.

If *option-value* is omitted, the specified option setting is deleted from the database. If it was a personal option setting, the value used reverts to the PUBLIC setting. If a TEMPORARY option is deleted, the option setting reverts to the permanent setting.

Note For all database options that accept integer values, Sybase IQ truncates any decimal *option-value* setting to an integer value. For example, the value 3.8 is truncated to 3.

The maximum length of *option-value* when set to a string is 127 bytes.

Warning! Changing option settings while fetching rows from a cursor is not supported, as it can lead to ill-defined behavior. For example, changing the DATE_FORMAT setting while fetching from a cursor returns different date formats among the rows in the result set. Do not change option settings while fetching rows.

Database options

For information about specific database options, see Chapter 2, “Database Options.”

Side effects

If TEMPORARY is not specified, an automatic commit is performed.

Standards

- **SQL92** Vendor extension.

- **Sybase** Not supported by Adaptive Server Enterprise. Sybase IQ does support some Adaptive Server Enterprise options using the SET statement.

Permissions

None required to set your own options. Must have DBA authority to set database options for another user or PUBLIC.

See also

Chapter 2, "Database Options"

SET OPTION statement [DBISQL]

Description

Changes DBISQL options.

Syntax

Syntax 1

SET [**TEMPORARY**] **OPTION**
... [*userid*. | **PUBLIC**.]*option-name* = [*option-value*]

Syntax 2

SET PERMANENT

Syntax 3

SET

Parameters

userid:
identifier, string or host-variable

option-name:
identifier, string, or host-variable

option-value:
host-variable (indicator allowed), string, identifier, or number

Usage

SET PERMANENT (Syntax 2) stores all current DBISQL options in the SYSOPTION system table. These settings are automatically established every time DBISQL is started for the current user ID.

Syntax 3 is used to display all of the current option settings. If there are temporary options set for DBISQL or the database server, these display; otherwise, permanent option settings are displayed.

If you incorrectly type the name of an option when you are setting the option, the incorrect name is saved in the SYSOPTION table. You can remove the incorrectly typed name from the SYSOPTION table by setting the option PUBLIC with an equality after the option name and no value:

```
SET OPTION PUBLIC.a_mistyped_name=;
```

See also Chapter 2, “Database Options”

SET SQLCA statement [ESQL]

Description Tells the SQL preprocessor to use a SQLCA other than the default global *sqlca*.

Syntax **SET SQLCA** *sqlca*

Parameters *sqlca*:
identifier or string

Examples Shows the owing function that can be found in a Windows DLL. Each application that uses the DLL has its own SQLCA.

```
an_sql_code FAR PASCAL ExecuteSQL( an_application *app,
char *com )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char *sqlcommand;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SET SQLCA "&app->.sqlca";
    sqlcommand = com;
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL EXECUTE IMMEDIATE :sqlcommand;
return( SQLCODE );
}
```

Usage The SET SQLCA statement tells the SQL preprocessor to use a SQLCA other than the default global *sqlca*. The *sqlca* must be an identifier or string that is a C language reference to a SQLCA pointer.

The current SQLCA pointer is implicitly passed to the database interface library on every Embedded SQL statement. All Embedded SQL statements that follow this statement in the C source file use the new SQLCA. This statement is necessary only when you are writing code that is reentrant. The *sqlca* should reference a local variable. Any global or module static variable is subject to being modified by another thread.

Side effects

None.

Standards
- **SQL92** Vendor extension.
- **Sybase** Not supported by Open Client/Open Server.

Permissions None.

See also "The Embedded SQL Interface" in the *Adaptive Server Anywhere Programming Interfaces Guide*

SIGNAL statement

Description Signals an exception condition.

Syntax **SIGNAL** *exception-name*

Usage SIGNAL lets you raise an exception. See Chapter 8, "Using Procedures and Batches" in the *Sybase IQ System Administration Guide* for a description of how exceptions are handled.

Side effects

None.

Standards
- **SQL92** Persistent Stored Module feature.
- **Sybase** SIGNAL is not supported by Adaptive Server Enterprise.

Permissions None.

See also BEGIN... END statement on page 422

RESIGNAL statement on page 620

START DATABASE statement [DBISQL]

Description Starts a database on the specified database server

Syntax **START DATABASE** *database-file*
... [**AS** *database-name*]
... [**ON** *engine-name*]
... [**AUTOSTOP** { **YES** | **NO** }]
... [**KEY** *key*]

Examples
- On a UNIX system, starts the database file */s1/sybase/sample_2.db* on the current server:

```
START DATABASE '/s1/sybase/sample_2.db'
```

- On a Windows system, starts the database file *c:\sybase\sample_2.db* as "sam2" on the server named "eng1":

```
START DATABASE 'c:\sybase\sample_2.db'
```

```
AS sam2
ON eng1
```

Usage

The database server must be running. The full path must be specified for the database file unless the file is located in the current directory.

The START DATABASE statement does not connect DBISQL to the specified database: a CONNECT statement must be issued to make a connection.

If *database-name* is not specified, a default name is assigned to the database. This default name is the root of the database file. For example, a database in file *c:\sybase\ASIQ-12_5\demo\asiqdemo.db* is given the default name asiqdemo.

If *engine-name* is not specified, the default database server is assumed. The default database server is the first started server among those currently running.

The default setting for the AUTOSTOP clause is YES. With AUTOSTOP set to YES, the database is unloaded when the last connection to it is dropped. If AUTOSTOP is set to NO, the database is not unloaded.

If the database is strongly encrypted, enter the KEY value (password) using the KEY clause.

Sybase recommends that you start only one database on a given Sybase IQ database server.

Side effects

None

Standards

- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions

Must have DBA authority.

START ENGINE statement [DBISQL]

Description

Starts a database server.

Syntax

START ENGINE AS *engine-name* [**STARTLINE** *command-string*]

Examples

- Start a database server, named “eng1”, without starting any databases on it:

  ```
  START ENGINE AS eng1
  ```

- The following example shows the use of a STARTLINE clause.

```
START ENGINE AS eng1 STARTLINE 'start_asiq -c 8096'
```

Usage

To specify a set of options for the server, use the STARTLINE keyword together with a command string. Valid command strings are those that conform to the database server command line description in Chapter 1, "Running the Database Server" in the *Sybase IQ Utility Guide*.

Note Several server options are required for Sybase IQ to operate well. To ensure that you are using the right set of options, Sybase recommends that you start your server by using either Sybase Central or a configuration file with the start_asiq command.

Side effects

None

Standards

- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions

None.

See also

STOP ENGINE statement [DBISQL] on page 656

Chapter 1, "Running the Database Server" in the *Sybase IQ Utility Guide*

START JAVA statement

Description

Starts the Java VM.

Syntax

START JAVA

Examples

Start the Java VM.

```
START JAVA
```

Usage

The main use of START JAVA is to load the VM at a convenient time so that when the user starts to use Java functionality there is no initial pause while the VM is loaded.

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions Must have DBA authority.

See also STOP JAVA statement on page 656

STOP DATABASE statement [DBISQL]

Description Stops a database on the specified database server.

Syntax **STOP DATABASE** *database-name*
... [**ON** *engine-name*]
... [**UNCONDITIONALLY**]

Examples Stop the database named "sample" on the default server:

```
STOP DATABASE sample
```

Usage If *engine-name* is not specified, all running engines are searched for a database of the specified name.

The *database-name* is the name specified in the -n parameter when the database is started, or in the DBN (DatabaseName) connection parameter. This name is typically the file name of the database file that holds the Catalog Store, without the *.db* extension, but can be any user-defined name

If UNCONDITIONALLY is supplied, the database is stopped even if there are connections to the database. If UNCONDITIONALLY is not specified, the database is not stopped if there are connections to it.

Side effects

None.

Standards
- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions Must have DBA authority.

See also DISCONNECT statement [DBISQL] on page 532

START DATABASE statement [DBISQL] on page 652

STOP ENGINE statement [DBISQL]

Description	Stops a database server
Syntax	**STOP ENGINE** *engine-name* [**UNCONDITIONALLY**]
Examples	Stop the database server named “sample”: `STOP ENGINE sample`
Usage	If UNCONDITIONALLY is supplied, the database server is stopped even if there are connections to the server. If UNCONDITIONALLY is not specified, the database server is not stopped if there are connections to it. Side effects None
Standards	• **SQL92** Vendor extension. • **Sybase** Not applicable.
Permissions	None
See also	START ENGINE statement [DBISQL] on page 653

STOP JAVA statement

Description	Stops the Java VM.
Syntax	**STOP JAVA**
Examples	Stops the Java VM: `STOP JAVA`
Usage	The main use of STOP JAVA is to economize on the use of system resources. Side effects None.
Standards	• **SQL92** Vendor extension. • **Sybase** Not applicable.
Permissions	DBA authority
See also	START JAVA statement on page 654

SYNCHRONIZE JOIN INDEX statement

Description

Synchronizes one or more join indexes after one of their base tables has been updated.

Syntax

SYNCHRONIZE JOIN INDEX [*join-index-name* [, *join-index-name*]...]

Examples

Synchronizes the join indexes emp_dept_join1 and emp_dept_join2:

```
SYNCHRONIZE JOIN INDEX emp_dept_join1, emp_dept_join2
```

Usage

When a base table that contributes to a join index is updated, Sybase IQ flags the join index as unavailable. Queries that previously took advantage of the join index perform an ad-hoc join instead, perhaps affecting their performance. The SYNCHRONIZE JOIN INDEX command lets you bring the join index up-to-date, making it available for queries to use.

Note A join index defines a one-to-many relationship (also known as primary key to foreign key) between two table columns. If an insert into the "one" (or primary key) column results in one or more duplicate values, the join index becomes invalid and cannot be synchronized. You must delete the rows containing the duplicate values before SYNCHRONIZE JOIN INDEX can make it valid again.

Synchronizing join indexes can be time-consuming, depending on the size of the base tables that make up the join. It is up to you to decide when to use this command. You can schedule it as a batch job at night or on weekends when you expect your system to have less work to do. You can perform it immediately after Sybase IQ commits a series of inserts and deletes to make the join index available as soon as possible. However, do not synchronize a join index after each insert or delete as the time to update the join index depends on the order of the updates to the tables.

SYNCHRONIZE JOIN INDEX lets you specify multiple *join-index-name*s, separated by commas. You must be the owner of each join index or the DBA. If you do not specify a *join-index-name*, Sybase IQ synchronizes all the join indexes you own (or all the join indexes in the database if you are the DBA), which might adversely affect the performance of your system.

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions

Must be owner of the join indexes or be DBA.

See also CREATE JOIN INDEX statement on page 481

TRIGGER EVENT statement

Description Triggers a named event. The event may be defined for event triggers or be a scheduled event.

Syntax **TRIGGER EVENT** *event-name* [(*parm* = *value*, ...)]

Usage Actions are tied to particular trigger conditions or schedules by a CREATE EVENT statement. You can use TRIGGER EVENT to force the event handler to execute, even when the scheduled time or trigger condition has not occurred. TRIGGER EVENT does not execute disabled event handlers.

parm = *value* When a triggering condition causes an event handler to execute, the database server can provide context information to the event handler using the event_parameter function. TRIGGER EVENT allows you to explicitly supply these parameters, to simulate a context for the event handler.

When you trigger an event, specify the event name. You can list event names by querying the system table SYSEVENT. For example:

```
SELECT event_id, event_name FROM SYS.SYSEVENT
```

Side effects

None.

Permissions Must have DBA authority.

See also ALTER EVENT statement on page 401

CREATE EVENT statement on page 458

Chapter 18, "Automating Tasks Using Schedules and Events" in the *Sybase IQ System Administration Guide*

TRUNCATE TABLE statement

Description Deletes all rows from a table without deleting the table definition.

Syntax **TRUNCATE TABLE** [*owner*.]*table-name*

Examples Deletes all rows from the department table:

```
TRUNCATE TABLE department
```

Usage

TRUNCATE TABLE is equivalent to a DELETE statement without a WHERE clause, except that each individual row deletion is not entered into the transaction log. After a TRUNCATE TABLE statement, the table structure and all of the indexes continue to exist until you issue a DROP TABLE statement. The column definitions and constraints remain intact, and permissions remain in effect.

The TRUNCATE TABLE statement is entered into the transaction log as a single statement, like data definition statements. Each deleted row is not entered into the transaction log.

Side effects

None.

Standards

- **SQL92** Transact-SQL extension.
- **Sybase** Supported by Adaptive Server Enterprise.

Permissions

- Must be the table owner or have DBA authority.
- For both temporary and base tables, you can execute TRUNCATE TABLE while other users have read access to the table. This behavior differs from Adaptive Server Anywhere, which requires exclusive access to truncate a base table. Sybase IQ table versioning ensures that TRUNCATE TABLE can occur while other users have read access; however, the version of the table these users see depends on when the read and write transactions commit.

See also

DELETE statement on page 525

Chapter 10, “Transactions and Versioning” in *Sybase IQ System Administration Guide*

UNION operation

Description

Combines the results of two or more select statements.

Syntax

select-without-order-by
... **UNION** [**ALL**] *select-without-order-by*
... [**UNION** [**ALL**] *select-without-order-by*]...
... [**ORDER BY** *integer* [**ASC** | **DESC**] [, ...]]

Examples

Lists all distinct surnames of employees and customers:

```
SELECT emp_lname
FROM employee
```

```
UNION
SELECT lname
FROM customer
```

Usage

The results of several SELECT statements can be combined into a larger result using UNION. The component SELECT statements must each have the same number of items in the select list, and cannot contain an ORDER BY clause.

The results of UNION ALL are the combined results of the component SELECT statements. The results of UNION are the same as UNION ALL except that duplicate rows are eliminated. Eliminating duplicates requires extra processing, so UNION ALL should be used instead of UNION where possible.

If corresponding items in two select lists have different data types, Sybase IQ chooses a data type for the corresponding column in the result, and automatically converts the columns in each component SELECT statement appropriately.

If ORDER BY is used, only integers are allowed in the order by list. These integers specify the position of the columns to be sorted.

The column names displayed are the same column names that display for the first SELECT statement.

Side effects

None.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Adaptive Server Enterprise, which also supports a COMPUTE clause.

Permissions

Must have SELECT permission for each of the component SELECT statements.

See also

SELECT statement on page 632

UPDATE statement

Description

Modifies existing rows of a single table, or a view that contains only one table.

Syntax

UPDATE *table*
... **SET** [*column-name = expression*, ...
... [**FROM** *table-expression*,]
... [**WHERE** *search-condition*]
... [**ORDER BY** *expression* [**ASC** | **DESC**] ,...]

FROM *table-expression*
table-expression:
table-spec | *table-expression join-type table-spec* [**ON** *condition*] | *table-expression*, ...

Examples

- Transfers employee Philip Chin (employee 129) from the sales department to the marketing department:

```
UPDATE employee
SET dept_id = 400
WHERE emp_id = 129 ;
```

- The Marketing Department (400) increases bonuses from 4% to 6% of each employee's base salary:

```
UPDATE employee
SET bonus = base * 6/100
WHERE dept_id=400;
```

- Each employee gets a pay increase with the department bonus:

```
UPDATE employee
SET emp.salary = emp.salary + dept.bonus
FROM employee emp, department dept
WHERE emp.deptnum = dept.deptnum;
```

- Another way to give each employee a pay increase with the department bonus:

```
UPDATE employee
SET emp.salary = emp.salary + dept.bonus
FROM employee emp JOIN department dept
ON emp.deptnum = dept.deptnum;
```

Usage

The table on which you use UPDATE may be a base table or a temporary table.

Note The base table cannot be part of any join index.

Each named column is set to the value of the expression on the right-hand side of the equal sign. Even *column-name* can be used in the expression—the old value is used.

The FROM clause can contain multiple tables with join conditions and returns all the columns from all the tables specified and filtered by the join condition and/or WHERE condition.

Using the wrong join condition in a FROM clause causes unpredictable results. If the FROM clause specifies a one-to-many join and the SET clause references a cell from the "many" side of the join, the cell is updated from the first value selected. In other words, if the join condition causes multiple rows of the table to be updated per row ID, the first row returned becomes the update result. For example:

```
UPDATE T1
SET T1.c2 = T2.c2
FROM T1 JOIN TO T2
ON T1.c1 = T2.c1
```

If table T2 has more than one row per T2.c1, results might be as follows:

```
T2.c1               T2.c2               T2.c3
1                   4                   3
1                   8                   1
1                   6                   4
1                   5                   2
```

With no ORDER BY clause, T1.c2 may be 4, 6, 8, or 9.

- With `ORDER BY T2.c3`, T1.c2 is updated to 8.
- With `ORDER BY T2.c3 DESC`, T1.c2 is updated to 6.

Sybase IQ rejects any UPDATE statement in which the table being updated is on the null-supplying side of an outer join. In other words:

- In a left outer join, the table on the left side of the join cannot be missing any rows on joined columns.
- In a right outer join, the table on the right side of the join cannot be missing any rows on joined columns.
- In a full outer join, neither table can be missing any rows on joined columns.

For example, in the following statement, table T1 is on the left side of a left outer join, and thus cannot contain be missing any rows:

```
UPDATE T1
```

```
SET T1.c2 = T2.c4
FROM T1 LEFT OUTER JOIN T2
ON T1.rowid = T2.rowid
```

Normally, the order in which rows are updated does not matter. However, in conjunction with the NUMBER(*) function, an ordering can be useful to get increasing numbers added to the rows in some specified order. If you are not using the NUMBER(*) function, avoid using the ORDER BY clause, because the UPDATE statement performs better without it.

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 do not increment sequentially due to row elimination. For more information about the NUMBER(*) function, see "NUMBER function [Miscellaneous]" on page 335.

You can use the ORDER BY clause to control the result from an UPDATE when the FROM clause contains multiple joined tables.

Sybase IQ ignores the ORDER BY clause in searched UPDATE and returns a message that the syntax is not valid ANSI syntax.

If no WHERE clause is specified, every row is updated. If you specify a WHERE clause, Sybase IQ updates only rows satisfying the search condition.

The left side of each SET clause must be a column in a base table.

Views can be updated provided the SELECT statement defining the view does not contain a GROUP BY clause or an aggregate function, or involve a UNION operation. The view should contain only one table.

Character strings inserted into tables are always stored in the case they are entered, regardless of whether the database is case sensitive or not. Thus a character data type column updated with a string Value is always held in the database with an upper-case V and the remainder of the letters lowercase. SELECT statements return the string as Value. If the database is not case sensitive, however, all comparisons make Value the same as value, VALUE, and so on. Further, if a single-column primary key already contains an entry Value, an INSERT of value is rejected, as it would make the primary key not unique.

If the update violates any check constraints, the whole statement is rolled back.

Sybase IQ supports scalar subqueries within the SET clause, for example:

```
UPDATE r
SET r.o= (SELECT MAX(t.o) FROM t ... WHERE t.y = r.y),
       r.s= (SELECT SUM(x.s) FROM x ... WHERE x.x = r.x)
WHERE r.a = 10
```

Sybase IQ supports DEFAULT column values in UPDATE statements. If a column has a DEFAULT value, this DEFAULT value is used as the value of the column in any UPDATE statement that does not explicitly modify the value for the column.

For detailed information on the use of column DEFAULT values, see "Using column defaults" in Chapter 9, "Ensuring Data Integrity" in the *Sybase IQ System Administration Guide*.

See the CREATE TABLE statement on page 499 for details about updating IDENTITY/AUTOINCREMENT columns, which are another type of DEFAULT column.

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** With the following exceptions, syntax of the IQ UPDATE statement is generally compatible with the Adaptive Server Enterprise UPDATE statement Syntax 1: Sybase IQ supports multiple tables with join conditions in the FROM clause.

  The Transact-SQL ROWCOUNT option has no effect on UPDATE operations in Sybase IQ.

  Updates of remote tables are limited to Sybase IQ syntax supported by CIS, as described in Chapter 17, "Server Classes for Remote Data Access" and Chapter 16, "Accessing Remote Data" in the *Sybase IQ System Administration Guide*.

Permissions

Must have UPDATE permission for the columns being modified.

UPDATE (positioned) statement [ESQL] [SP]

Description

Modifies the data at the current location of a cursor.

Syntax

UPDATE *table-list*
SET *set-item*, …
WHERE CURRENT OF *cursor-name*

Parameters

cursor-name:
identifier | hostvar

set-item:
column-name [.*field-name*…] = *scalar-value*)

SET
The columns that are referenced in *set-item* must be in the base table that is updated. They cannot refer to aliases, nor to columns from other tables or views. If the table you are updating is given a correlation name in the cursor specification, you must use the correlation name in the SET clause.

The expression on the right side of the SET clause may reference columns, constants, variables, and expressions from the SELECT clause of the query. The *set-item* expression cannot contain functions or expressions.

Examples

The following is an example of an UPDATE statement WHERE CURRENT OF cursor:

```
UPDATE employee SET emp_lname = 'Jones'
WHERE CURRENT OF emp_cursor;
```

Usage

This form of the UPDATE statement updates the current row of the specified cursor. The current row is defined to be the last row successfully fetched from the cursor, and the last operation on the cursor cannot have been a positioned DELETE statement.

The requested columns are set to the specified values for the row at the current row of the specified query. The columns must be in the select list of the specified open cursor.

Changes effected by positioned UPDATE statements are visible in the cursor result set, except where client-side caching prevents seeing these changes. Rows that are updated so that they no longer meet the requirements of the WHERE clause of the open cursor are still visible.

Sybase does not recommend the use of ORDER BY in the WHERE CURRENT OF clause. The ORDER BY columns may be updated, but the result set does not reorder. The results appear to fetch out of order and appear to be incorrect.

Since Sybase IQ does not support the CREATE VIEW... WITH CHECK OPTION, positioned UPDATE does not support this option. The WITH CHECK OPTION does not allow an update that creates a row that is not visible by the view.

A rowid column cannot be updated by a positioned UPDATE.

Sybase IQ supports repeatedly updating the same row in the result set.

Standards

- **SQL92** Entry-level feature. The range of cursors that can be updated may contain vendor extensions if the ANSI_UPDATE_CONSTRAINTS option is set to OFF.

- **SQL99** Core feature. The range of cursors that can be updated may contain vendor extensions if the ANSI_UPDATE_CONSTRAINTS option is set to OFF.
- **Sybase** Embedded SQL use is supported by Open Client/Open Server, and procedure and trigger use is supported in Adaptive Server Anywhere.

Permissions

Must have UPDATE permission on the columns being modified.

See also

DECLARE CURSOR statement [ESQL] [SP] on page 516

DELETE statement on page 525

DELETE (positioned) statement [ESQL] [SP] on page 527

UPDATE statement on page 661

"sp_iqcursorinfo procedure" on page 759

WAITFOR statement

Description

Delays processing for the current connection for a specified amount of time or until a given time.

Syntax

WAITFOR { **DELAY** *time* | **TIME** *time* }

time: *string*

Examples

Example 1 The following example waits for three seconds:

```
WAITFOR DELAY '00:00:03'
```

Example 2 The following example waits for 0.5 seconds (500 milliseconds):

```
WAITFOR DELAY '00:00:00:500'
```

Example 3 The following example waits until 8 p.m.:

```
WAITFOR TIME '20:00'
```

Usage

If DELAY is used, processing is suspended for the given interval. If TIME is specified, processing is suspended until the server time reaches the time specified.

If the current server time is greater than the time specified, processing is suspended until that time on the following day.

WAITFOR provides an alternative to the following statement, and might be useful for customers who choose not to enable Java in the database:

```
call java.lang.Thread.sleep(
<time_to_wait_in_millisecs> )
```

In many cases, scheduled events are a better choice than using WAITFOR TIME, because scheduled events execute on their own connection.

Side effects

The implementation of this statement uses a worker thread while it is waiting. This uses up one of the threads specified by the -gn server command line option.

Standards

- **SQL92** Vendor extension.
- **SQL99** Vendor extension.
- **Sybase** This statement is also implemented by Adaptive Server Enterprise.

Permissions

None.

See also

CREATE EVENT statement on page 458

WHENEVER statement [ESQL]

Description

Specifies error handling in an Embedded SQL program.

Syntax

WHENEVER
{ **SQLERROR** | **SQLWARNING** | **NOTFOUND** }
... {**GOTO** *label* | **STOP** | **CONTINUE** | *C code;* }

Parameters

label:
identifier

Examples

The following are examples of the WHENEVER statement:

```
EXEC SQL WHENEVER NOTFOUND GOTO done;
EXEC SQL WHENEVER SQLERROR
    {
       PrintError( &sqlca );
       return( FALSE );
    };
```

Usage

The WHENEVER statement is used to trap errors, warnings, and exceptional conditions encountered by the database when processing SQL statements. The statement can be put anywhere in an Embedded SQL C program, and does not generate any code. The preprocessor generates code following each successive SQL statement. The error action remains in effect for all Embedded SQL statements from the source line of the WHENEVER statement until the next WHENEVER statement with the same error condition, or the end of the source file.

Note The error conditions are in effect based on positioning in the C language source file and not on when the statements are executed.

The default action is CONTINUE.

WHENEVER is provided for convenience in simple programs. Most of the time, checking the sqlcode field of the SQLCA (SQLCODE) directly is the easiest way to check error conditions. In this case, WHENEVER would not be used. If fact, all the WHENEVER statement does is cause the preprocessor to generate an *if (SQLCODE)* test after each statement.

Side effects

None.

Standards

- **SQL92** Entry-level feature.
- **Sybase** Supported by Open Client/Open Server.

Permissions

None.

WHILE statement [T-SQL]

Description

Provides repeated execution of a statement or compound statement.

Syntax

WHILE *expression*
... statement

Examples

Illustrates the use of WHILE:

```
WHILE (SELECT AVG(unit_price) FROM product) < $30
BEGIN
    DELETE FROM product
    WHERE unit_price = MAX(unit_price)
    IF ( SELECT MAX(unit_price) FROM product ) < $50
       BREAK
```

```
END
```

The BREAK statement breaks the WHILE loop if the most expensive product has a price less than $50. Otherwise the loop continues until the average price is greater than $30.

Usage

The WHILE conditional affects the performance of only a single SQL statement, unless statements are grouped into a compound statement between the keywords BEGIN and END.

The BREAK statement and CONTINUE statement can be used to control execution of the statements in the compound statement. The BREAK statement terminates the loop, and execution resumes after the END keyword, marking the end of the loop. The CONTINUE statement causes the WHILE loop to restart, skipping any statements after the CONTINUE.

Side effects

None.

Standards

- **SQL92** Transact-SQL extension.
- **Sybase** Supported by Adaptive Server Enterprise.

Permissions

None

CHAPTER 7

Differences from Other SQL Dialects

About this chapter

Sybase IQ conforms to the ANSI SQL89 standard but has many additional features defined in the IBM DB2 and SAA specification, 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 and 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 5, “SQL Functions” for a description of these.

Integrity

Adaptive Server 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 will then ensure referential integrity for these columns - whenever these columns are modified or a row is inserted into this table, these columns will be 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

Adaptive Server 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 will 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
```

The MODIFY can be used to change the maximum length of a character column as well as converting from one data type to another. For more information, see ALTER TABLE statement.

Subqueries not always allowed

Unlike Adaptive Server 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. It does not support Adaptive Server Anywhere's extensions.

Many SQL implementations only allow subqueries on the right side of a comparison operator. For example, the following command is valid in Adaptive Server IQ but not valid in most other SQL implementations.

```
SELECT emp_lname,
          emp_birthdate,
          (  SELECT skill
             FROM department
             WHERE emp_id = employee.emp_ID
             AND dept_id = 200 )
FROM employee
```

Additional functions

Sybase IQ supports several functions not in the ANSI SQL definition. See Chapter 5, “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 8

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 8-1 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 8-1: Sybase IQ database object size and number limitations

Item	Limitation
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
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.
Number of columns per table	Sybase IQ supports up to 45,000 columns in a table. There might be performance penalties with more than 10,000 columns in a table.
Number of files per database	Operating system limit that user can adjust; for example, using NOFILE. Typically, 2047 files per database.
Number of rows per table	Limited by table size, upper limit 2^{48}
Number of rows per LOAD TABLE or INSERT statement	2GB - 1
Number of tables per database	4,293,918,719
Number of tables referenced per transaction	No limit
Number of tables or views referenced per query	512
Number of tables or views in a single FROM clause	16 to 64, depending on the query, with join optimizer turned on.
Number of UNION branches per query	128. If each branch has multiple tables in the FROM clause, the limit on tables per query reduces the number of UNION branches allowed.
Table size	Limited by database size.
Row size	Sybase recommends a limit of half the page size.
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

Item	Limitation
Maximum length of SQL statement	Defaults to Catalog page size (not IQ page size) of connected database.This affects long commands such as RESTORE statements with many renamed databases. To run such commands, you may start the server with an increased -gp setting, although the default of -gp 4096 should generally be used.
Maximum length of variable-length FILLER column	512 bytes
Number of indexes	32,767 per table
Maximum key size	255 bytes for single-column index 5300 bytes for multicolumn index
Number of tables per join index (number of tables that can be joined in one query block)	32
Number of values in an IN list	250,000
Number of stored procedures per database	2^32 – 1 = 4 294 967 295
Number of events per database	2^31 – 1 = 2 147 483 647
IQ page size	Must be between 64KB and 512KB.
Maximum number of users (connected and concurrent)	1000 on 64-bit platforms: AIX, Sun Solaris, and HP 200 on 32-bit platforms: Linux and Windows.
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

Sun Solaris OS error initializing raw device

Sun Solaris only: When creating a database or dbspace on a raw device in version 12.6 and later releases, Sybase IQ performs a series of calculations to determine the correct size of the raw partition. Each time Sybase IQ tries to initialize the device using its calculation, an operating system error is reported until an appropriate size is calculated. The database or dbspace is successfully created and the errors can be ignored. These errors were not reported in Sybase IQ version 12.5 and earlier releases when creating a database or dbspace on a raw device.

CHAPTER 9

System Tables

About this chapter

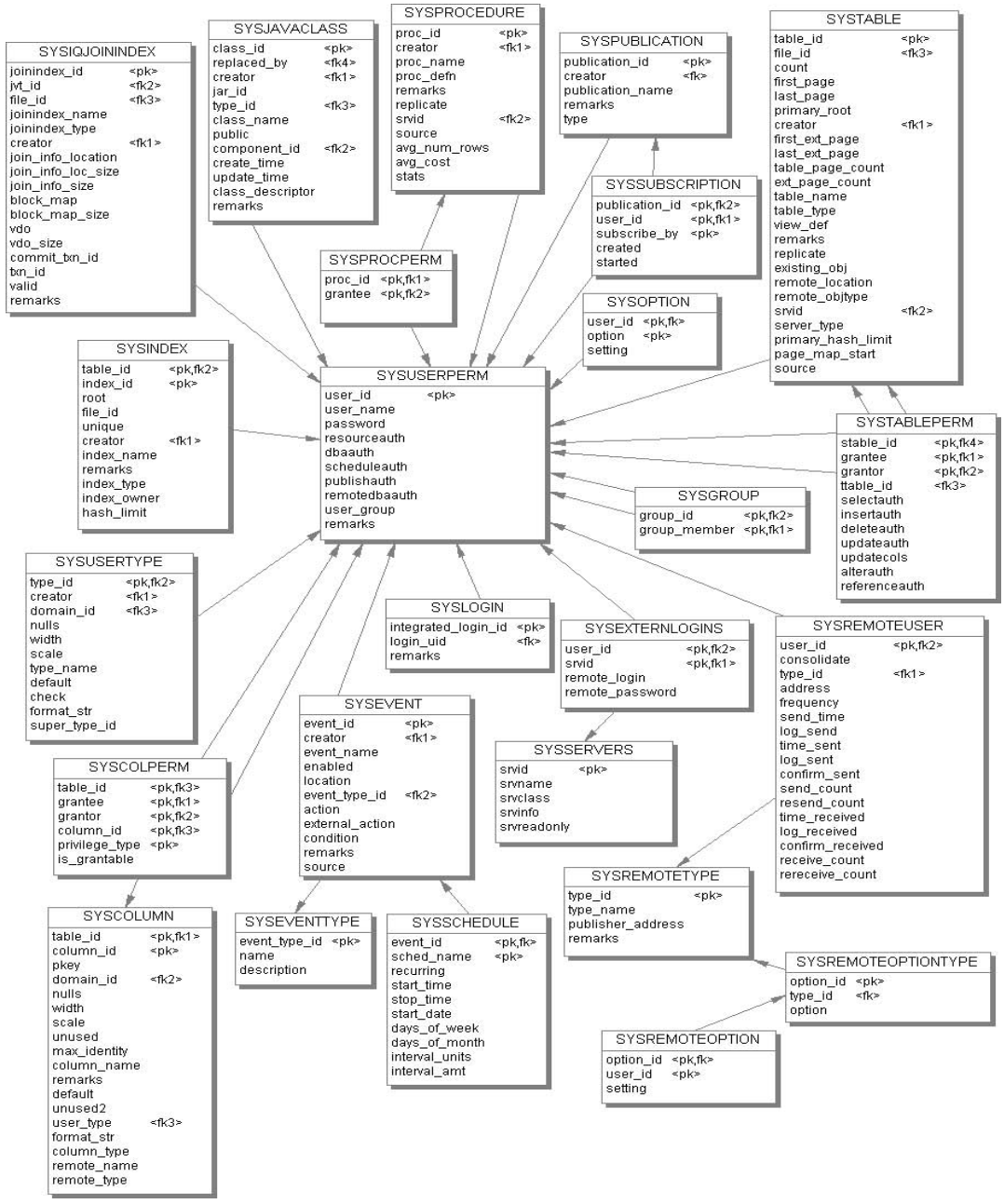
The structure of every Sybase IQ database is described in a number of system tables. The Entity-Relationship diagrams at the start of this chapter show all the system tables and the foreign keys connecting the system tables.

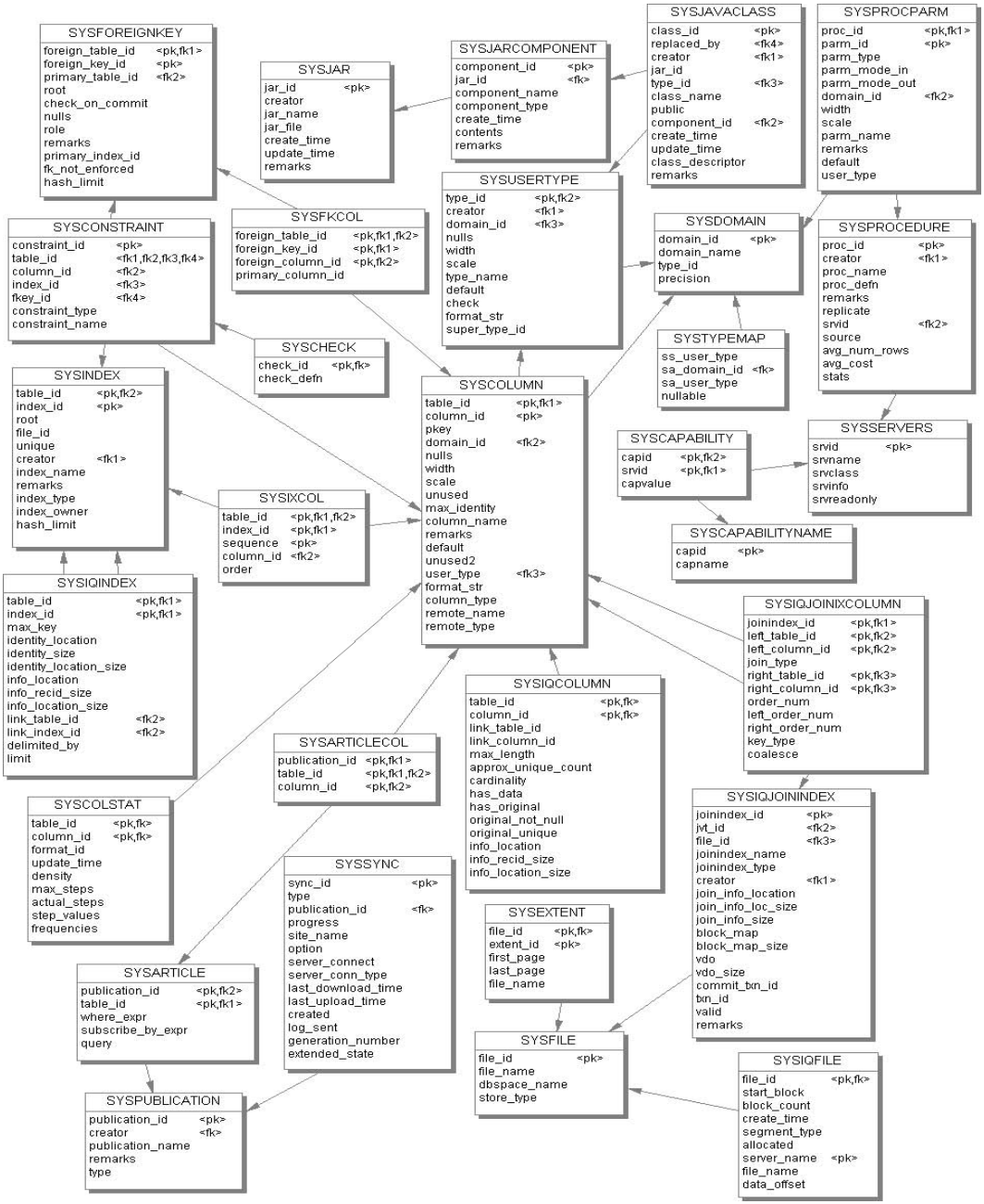
System tables are owned by the SYS user ID. The contents of these tables can be changed only by the database system. Thus, you cannot use the UPDATE, DELETE, and INSERT commands to modify the contents of these tables. Further, you cannot change the structure of these tables using the ALTER TABLE and DROP commands.

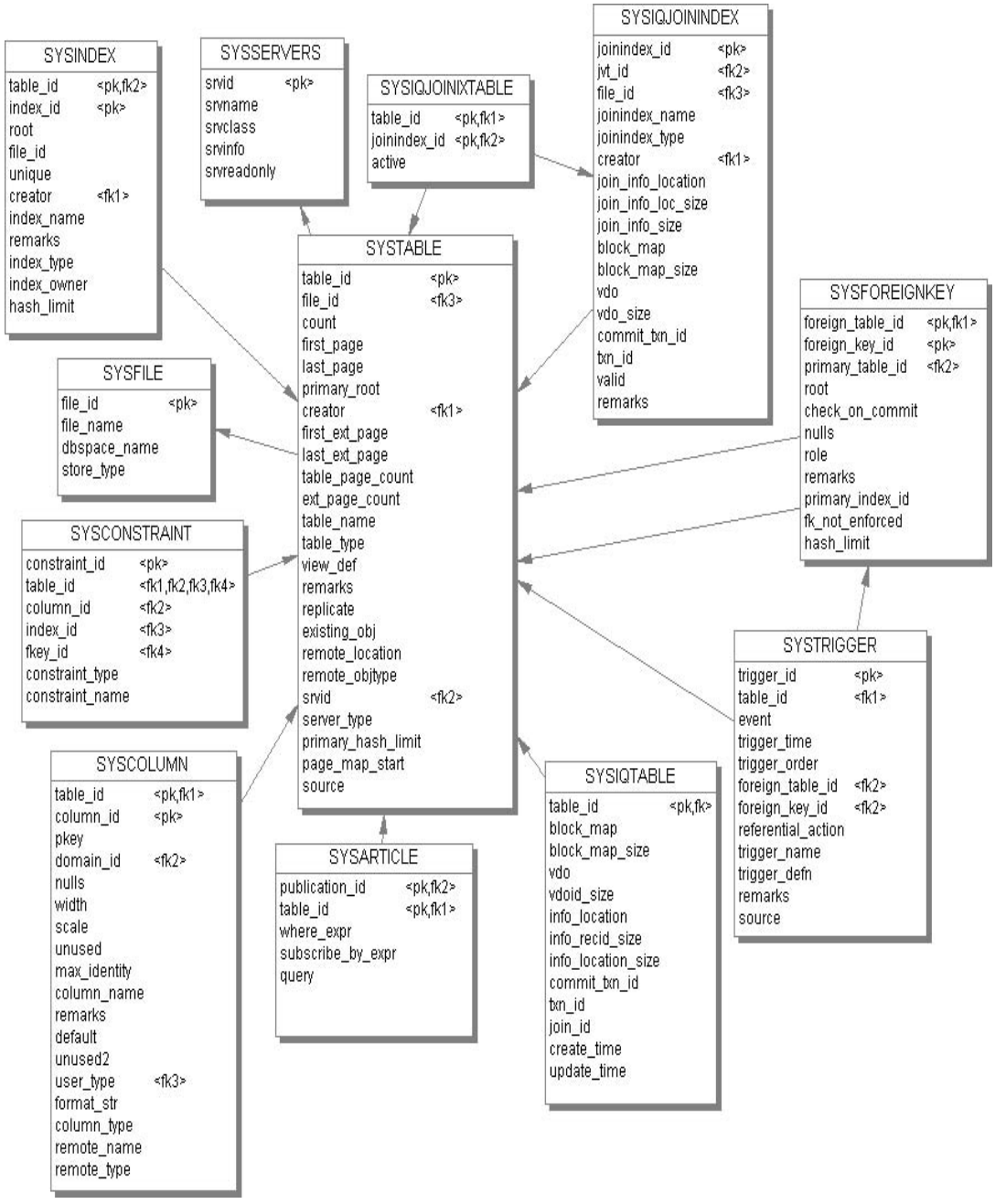
Contents

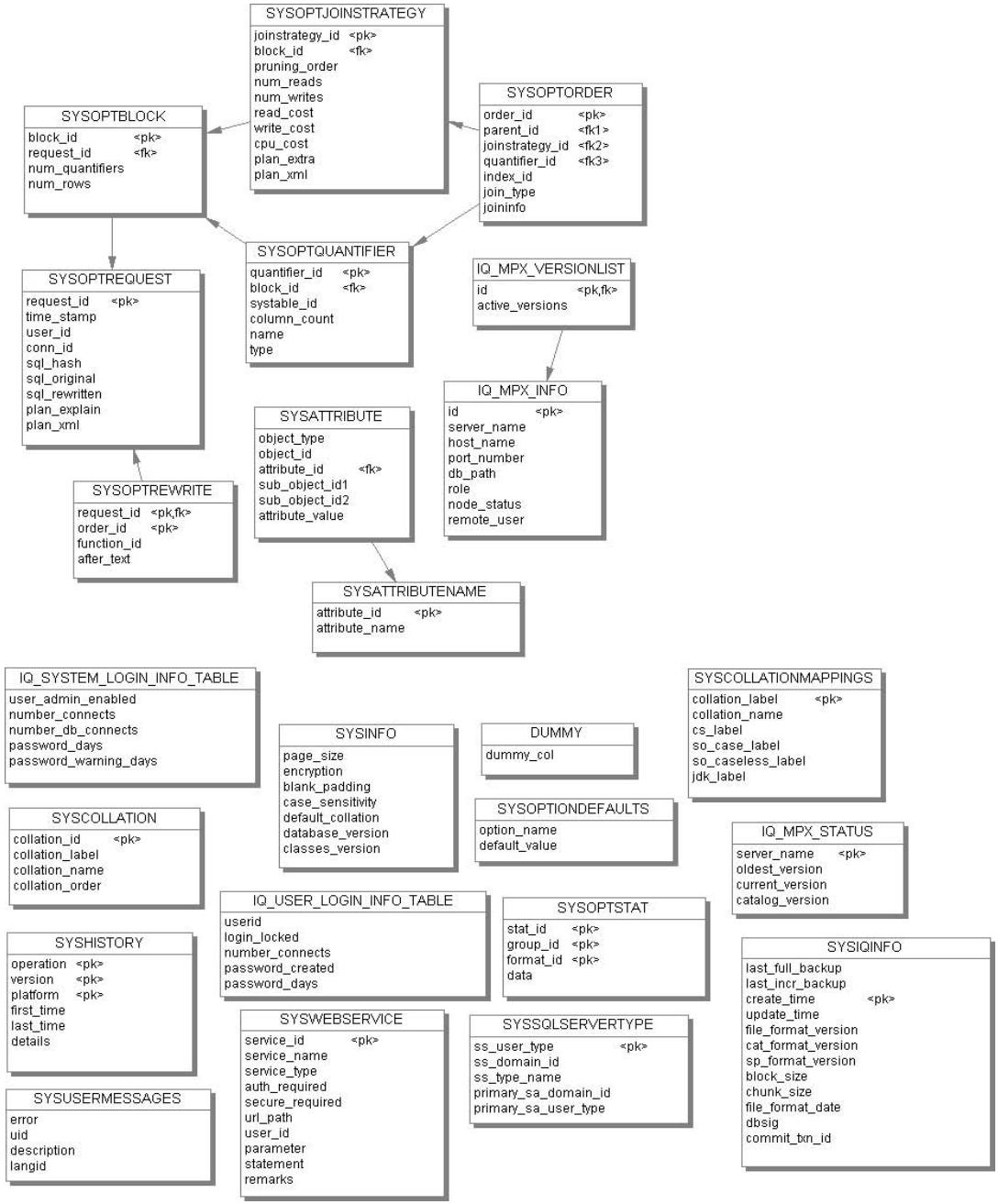
System tables diagrams

The Sybase IQ system tables are shown in the following set of diagrams. Foreign-key relations between tables are indicated by arrows; arrows lead from the foreign table to the primary table.


SYSIQJOININDEX
joinindex_id <pk>
jvt_id <fk2>
file_id <fk3>
joinindex_name
joinindex_type
creator <fk1>
join_info_location
join_info_loc_size
join_info_size
block_map
block_map_size
vdo
vdo_size
commit_txn_id
txn_id
valid
remarks
SYSJAVACLASS
class_id <pk>
replaced_by <fk4>
creator <fk1>
jar_id
type_id <fk3>
class_name
public
component_id <fk2>
create_time
update_time
class_descriptor
remarks
SYSPROCEDURE
proc_id <pk>
creator <fk1>
proc_name
proc_defn
remarks
replicate
srvid <fk2>
source
avg_num_rows
avg_cost
stats
SYSPUBLICATION
publication_id <pk>
creator <fk>
publication_name
remarks
type
SYSTABLE
table_id <pk>
file_id <fk3>
count
first_page
last_page
primary_root
creator <fk1>
first_ext_page
last_ext_page
table_page_count
ext_page_count
table_name
table_type
view_def
remarks
replicate
existing_obj
remote_location
remote_objtype
srvid <fk2>
server_type
primary_hash_limit
page_map_start
source
SYSSUBSCRIPTION
publication_id <pk,fk2>
user_id <pk,fk1>
subscribe_by <pk>
created
started
SYSPROCPERM
proc_id <pk,fk1>
grantee <pk,fk2>
SYSOPTION
user_id <pk,fk>
option <pk>
setting
SYSINDEX
table_id <pk,fk2>
index_id <pk>
root
file_id
unique
creator <fk1>
index_name
remarks
index_type
index_owner
hash_limit
SYSUSERPERM
user_id <pk>
user_name
password
resourceauth
dbaauth
scheduleauth
publishauth
remotedbaauth
user_group
remarks
SYSTABLEPERM
stable_id <pk,fk4>
grantee <pk,fk1>
grantor <pk,fk2>
ttable_id <fk3>
selectauth
insertauth
deleteauth
updateauth
updatecols
alterauth
referenceauth
SYSGROUP
group_id <pk,fk2>
group_member <pk,fk1>
SYSUSERTYPE
type_id <pk,fk2>
creator <fk1>
domain_id <fk3>
nulls
width
scale
type_name
default
check
format_str
super_type_id
SYSLOGIN
integrated_login_id <pk>
login_uid <fk>
remarks
SYSEXTERNLOGINS
user_id <pk,fk2>
srvid <pk,fk1>
remote_login
remote_password
SYSREMOTEUSER
user_id <pk,fk2>
consolidate
type_id <fk1>
address
frequency
send_time
log_send
time_sent
log_sent
confirm_sent
send_count
resend_count
time_received
log_received
confirm_received
receive_count
rereceive_count
SYSEVENT
event_id <pk>
creator <fk1>
event_name
enabled
location
event_type_id <fk2>
action
external_action
condition
remarks
source
SYSSERVERS
srvid <pk>
srvname
srvclass
srvinfo
srvreadonly
SYSCOLPERM
table_id <pk,fk3>
grantee <pk,fk1>
grantor <pk,fk2>
column_id <pk,fk3>
privilege_type <pk>
is_grantable
SYSREMOTETYPE
type_id <pk>
type_name
publisher_address
remarks
SYSREMOTEOPTIONTYPE
option_id <pk>
type_id <fk>
option
SYSCOLUMN
table_id <pk,fk1>
column_id <pk>
pkey
domain_id <fk2>
nulls
width
scale
unused
max_identity
column_name
remarks
default
unused2
user_type <fk3>
format_str
column_type
remote_name
remote_type
SYSEVENTTYPE
event_type_id <pk>
name
description
SYSSCHEDULE
event_id <pk,fk>
sched_name <pk>
recurring
start_time
stop_time
start_date
days_of_week
days_of_month
interval_units
interval_amt
SYSREMOTEOPTION
option_id <pk,fk>
user_id <pk>
setting


SYSFOREIGNKEY
foreign_table_id <pk,fk1>
foreign_key_id <pk>
primary_table_id <fk2>
root
check_on_commit
nulls
role
remarks
primary_index_id
fk_not_enforced
hash_limit
SYSJAR
jar_id <pk>
creator
jar_name
jar_file
create_time
update_time
remarks
SYSJARCOMPONENT
component_id <pk>
jar_id <fk>
component_name
component_type
create_time
contents
remarks
SYSJAVACLASS
class_id <pk>
replaced_by <fk4>
creator <fk1>
jar_id
type_id <fk3>
class_name
public
component_id <fk2>
create_time
update_time
class_descriptor
remarks
SYSPROCPARM
proc_id <pk,fk1>
parm_id <pk>
parm_type
parm_mode_in
parm_mode_out
domain_id <fk2>
width
scale
parm_name
remarks
default
user_type
SYSUSERTYPE
type_id <pk,fk2>
creator <fk1>
domain_id <fk3>
nulls
width
scale
type_name
default
check
format_str
super_type_id
SYSDOMAIN
domain_id <pk>
domain_name
type_id
precision
SYSPROCEDURE
proc_id <pk>
creator <fk1>
proc_name
proc_defn
remarks
replicate
srvid <fk2>
source
avg_num_rows
avg_cost
stats
SYSFKCOL
foreign_table_id <pk,fk1,fk2>
foreign_key_id <pk,fk1>
foreign_column_id <pk,fk2>
primary_column_id
SYSCONSTRAINT
constraint_id <pk>
table_id <fk1,fk2,fk3,fk4>
column_id <fk2>
index_id <fk3>
fkey_id <fk4>
constraint_type
constraint_name
SYSCHECK
check_id <pk,fk>
check_defn
SYSTYPEMAP
ss_user_type
sa_domain_id <fk>
sa_user_type
nullable
SYSINDEX
table_id <pk,fk2>
index_id <pk>
root
file_id
unique
creator <fk1>
index_name
remarks
index_type
index_owner
hash_limit
SYSCOLUMN
table_id <pk,fk1>
column_id <pk>
pkey
domain_id <fk2>
nulls
width
scale
unused
max_identity
column_name
remarks
default
unused2
user_type <fk3>
format_str
column_type
remote_name
remote_type
SYSCAPABILITY
capid <pk,fk2>
srvid <pk,fk1>
capvalue
SYSSERVERS
srvid <pk>
srvname
srvclass
srvinfo
srvreadonly
SYSIXCOL
table_id <pk,fk1,fk2>
index_id <pk,fk1>
sequence <pk>
column_id <fk2>
order
SYSCAPABILITYNAME
capid <pk>
capname
SYSIQINDEX
table_id <pk,fk1>
index_id <pk,fk1>
max_key
identity_location
identity_size
identity_location_size
info_location
info_recid_size
info_location_size
link_table_id <fk2>
link_index_id <fk2>
delimited_by
limit
SYSIQJOINIXCOLUMN
joinindex_id <pk,fk1>
left_table_id <pk,fk2>
left_column_id <pk,fk2>
join_type
right_table_id <pk,fk3>
right_column_id <pk,fk3>
order_num
left_order_num
right_order_num
key_type
coalesce
SYSIQCOLUMN
table_id <pk,fk>
column_id <pk,fk>
link_table_id
link_column_id
max_length
approx_unique_count
cardinality
has_data
has_original
original_not_null
original_unique
info_location
info_recid_size
info_location_size
SYSARTICLECOL
publication_id <pk,fk1>
table_id <pk,fk1,fk2>
column_id <pk,fk2>
SYSCOLSTAT
table_id <pk,fk>
column_id <pk,fk>
format_id
update_time
density
max_steps
actual_steps
step_values
frequencies
SYSIQJOININDEX
joinindex_id <pk>
jvt_id <fk2>
file_id <fk3>
joinindex_name
joinindex_type
creator <fk1>
join_info_location
join_info_loc_size
join_info_size
block_map
block_map_size
vdo
vdo_size
commit_txn_id
txn_id
valid
remarks
SYSSYNC
sync_id <pk>
type
publication_id <fk>
progress
site_name
option
server_connect
server_conn_type
last_download_time
last_upload_time
created
log_sent
generation_number
extended_state
SYSEXTENT
file_id <pk,fk>
extent_id <pk>
first_page
last_page
file_name
SYSARTICLE
publication_id <pk,fk2>
table_id <pk,fk1>
where_expr
subscribe_by_expr
query
SYSFILE
file_id <pk>
file_name
dbspace_name
store_type
SYSIQFILE
file_id <pk,fk>
start_block
block_count
create_time
segment_type
allocated
server_name <pk>
file_name
data_offset
SYSPUBLICATION
publication_id <pk>
creator <fk>
publication_name
remarks
type


SYSINDEX
table_id <pk,fk2>
index_id <pk>
root
file_id
unique
creator <fk1>
index_name
remarks
index_type
index_owner
hash_limit
SYSSERVERS
srvid <pk>
srvname
srvclass
srvinfo
srvreadonly
SYSIQJOINIXTABLE
table_id <pk,fk1>
joinindex_id <pk,fk2>
active
SYSIQJOININDEX
joinindex_id <pk>
jvt_id <fk2>
file_id <fk3>
joinindex_name
joinindex_type
creator <fk1>
join_info_location
join_info_loc_size
join_info_size
block_map
block_map_size
vdo
vdo_size
commit_txn_id
txn_id
valid
remarks
SYSTABLE
table_id <pk>
file_id <fk3>
count
first_page
last_page
primary_root
creator <fk1>
first_ext_page
last_ext_page
table_page_count
ext_page_count
table_name
table_type
view_def
remarks
replicate
existing_obj
remote_location
remote_objtype
srvid <fk2>
server_type
primary_hash_limit
page_map_start
source
SYSFOREIGNKEY
foreign_table_id <pk,fk1>
foreign_key_id <pk>
primary_table_id <fk2>
root
check_on_commit
nulls
role
remarks
primary_index_id
fk_not_enforced
hash_limit
SYSFILE
file_id <pk>
file_name
dbspace_name
store_type
SYSCONSTRAINT
constraint_id <pk>
table_id <fk1,fk2,fk3,fk4>
column_id <fk2>
index_id <fk3>
fkey_id <fk4>
constraint_type
constraint_name
SYSTRIGGER
trigger_id <pk>
table_id <fk1>
event
trigger_time
trigger_order
foreign_table_id <fk2>
foreign_key_id <fk2>
referential_action
trigger_name
trigger_defn
remarks
source
SYSCOLUMN
table_id <pk,fk1>
column_id <pk>
pkey
domain_id <fk2>
nulls
width
scale
unused
max_identity
column_name
remarks
default
unused2
user_type <fk3>
format_str
column_type
remote_name
remote_type
SYSARTICLE
publication_id <pk,fk2>
table_id <pk,fk1>
where_expr
subscribe_by_expr
query
SYSIQTABLE
table_id <pk,fk>
block_map
block_map_size
vdo
vdoid_size
info_location
info_recid_size
info_location_size
commit_txn_id
txn_id
join_id
create_time
update_time


SYSOPTJOINSTRATEGY
joinstrategy_id <pk>
block_id <fk>
pruning_order
num_reads
num_writes
read_cost
write_cost
cpu_cost
plan_extra
plan_xml
SYSOPTBLOCK
block_id <pk>
request_id <fk>
num_quantifiers
num_rows
SYSOPTORDER
order_id <pk>
parent_id <fk1>
joinstrategy_id <fk2>
quantifier_id <fk3>
index_id
join_type
joininfo
SYSOPTQUANTIFIER
quantifier_id <pk>
block_id <fk>
systable_id
column_count
name
type
SYSOPTREQUEST
request_id <pk>
time_stamp
user_id
conn_id
sql_hash
sql_original
sql_rewritten
plan_explain
plan_xml
IQ_MPX_VERSIONLIST
id <pk,fk>
active_versions
IQ_MPX_INFO
id <pk>
server_name
host_name
port_number
db_path
role
node_status
remote_user
SYSATTRIBUTE
object_type
object_id
attribute_id <fk>
sub_object_id1
sub_object_id2
attribute_value
SYSOPTREWRITE
request_id <pk,fk>
order_id <pk>
function_id
after_text
SYSATTRIBUTENAME
attribute_id <pk>
attribute_name
IQ_SYSTEM_LOGIN_INFO_TABLE
user_admin_enabled
number_connects
number_db_connects
password_days
password_warning_days
SYSINFO
page_size
encryption
blank_padding
case_sensitivity
default_collation
database_version
classes_version
DUMMY
dummy_col
SYSCOLLATIONMAPPINGS
collation_label <pk>
collation_name
cs_label
so_case_label
so_caseless_label
jdk_label
SYSCOLLATION
collation_id <pk>
collation_label
collation_name
collation_order
SYSOPTIONDEFAULTS
option_name
default_value
IQ_MPX_STATUS
server_name <pk>
oldest_version
current_version
catalog_version
IQ_USER_LOGIN_INFO_TABLE
userid
login_locked
number_connects
password_created
password_days
SYSOPTSTAT
stat_id <pk>
group_id <pk>
format_id <pk>
data
SYSHISTORY
operation <pk>
version <pk>
platform <pk>
first_time
last_time
details
SYSIQINFO
last_full_backup
last_incr_backup
create_time <pk>
update_time
file_format_version
cat_format_version
sp_format_version
block_size
chunk_size
file_format_date
dbsig
commit_txn_id
SYSWEBSERVICE
service_id <pk>
service_name
service_type
auth_required
secure_required
url_path
user_id
parameter
statement
remarks
SYSSQLSERVERTYPE
ss_user_type <pk>
ss_domain_id
ss_type_name
primary_sa_domain_id
primary_sa_user_type
SYSUSERMESSAGES
error
uid
description
langid

System tables descriptions

This section contains a description of each of the system tables. The system tables are described via the CREATE TABLE commands used to create them. They serve as good examples of how tables are created in SQL. Following the CREATE TABLE command, each column is briefly described.

Several of the columns have only two possible values. Usually these values are "Y" and "N" for "yes" and "no," respectively. These columns are designated by "(Y/N)".

DUMMY system table

```
CREATE TABLE SYS.DUMMY (
    dummy_col INT NOT NULL
)
```

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
```

dummy_col Not used. It is present because a table with no columns cannot be created.

The use of the DUMMY system table is implied for all queries that do not have a FROM clause; for example, `SELECT NOW();`.

These queries are run by Adaptive Server Anywhere (the catalog engine), rather than by Sybase IQ. You can create a dummy table in the Sybase IQ database, for example:

```
CREATE TABLE iq_dummy (dummy_col INT NOT NULL);
```

and use this table explicitly:

```
SELECT NOW() FROM iq_dummy;
```

For more information, see FROM clause in Chapter 6, "SQL Statements."

IQ_MPX_INFO system table

```
CREATE TABLE "DBA".IQ_MPX_INFO (
      id          numeric(8,0) IDENTITY NOT NULL,
      server_name varchar(30) NOT NULL,
      host_name   varchar(40) NOT NULL,
      port_number numeric(8,0)NOT NULL,
      db_path varchar(1024) NOT NULL,
      role        char(1) NOT NULL,
      node_status varchar(10) NOT NULL,
      remote_user varchar(40) NOT NULL,
      PRIMARY KEY (id),
      UNIQUE      (server_name)
    )IN SYSTEM;
```

Each row in this table contains information about a particular server in a multiplex. The table contains only one row per server.

Sybase Central uses the information in this table to manage the multiplex. The information is also used internally in the Sybase IQ procedures, triggers, and events. SQL Remote automatically replicates changes in this table between servers.

The information in this table changes only when:

- Query servers are added or removed from the multiplex
- A query server replaces the write server
- Query servers are excluded from or included in the multiplex environment

UPDATE permission is not granted to PUBLIC.

SQL Remote automatically replicates changes in this table between multiplex servers.

id The sequence number of this server.

server_name The server name of this server.

host_name The host on which this server runs.

port_number The TCP port number on which to start this server.

db_path The full path to the Catalog Store (.db file) for this server. Each server's database files must be placed on a file system local to its node. Each server may use a different name for its catalog database file (and database name).

role Write server ('W') or query server ('R').

node_status Server is Active or Inactive.

remote_user The remote user ID for SQL Remote replication at this server.

IQ_MPX_STATUS system table

```
CREATE TABLE "DBA".IQ_MPX_STATUS (
     server_name     varchar(30)  NOT NULL,
     oldest_version  unsigned bigint NOT NULL,
     current_version varchar(10)     NOT NULL,
     catalog_version unsigned bigint NOT NULL,
    )
```

This table contains dynamic information to support management of old table versions in a multiplex environment. The information in this table changes as transactions begin or commit on servers of the multiplex.
Each server updates its own row in the table automatically on a transaction-commit and begin-transaction. SQL Remote propagates changes between the multiplex servers. UPDATE permission is not granted to PUBLIC.

server_name The server name of this server.

oldest_version The version number of the oldest active transaction on this server.

current_version The current transaction number on this server.

catalog_version Each time the write server modifies the table schema, the catalog version advances to the current version. Each time you synchronize query servers, the catalog version at a query server advances to the write server's version, but remains static until the next synchronization.

By examining the catalog_version column, you can see whether query servers are synchronized.

IQ_MPX_VERSIONLIST system table

```
CREATE TABLE "DBA".IQ_MPX_VERSIONLIST (
      id integer PRIMARY KEY,
      active_versions long varchar
)IN SYSTEM;
```

This table contains one row for each query server in the multiplex (including inactive ones).

id The identifier for the query server from server name of this server.

active_versions The set of active versions.

IQ_SYSTEM_LOGIN_INFO_TABLE system table

```
CREATE TABLE DBA.IQ_SYSTEM_LOGIN_INFO_TABLE (
  user_admin_enabled   CHAR(1) NOT NULL,
  number_connects   INT NOT NULL,
  number_db_connects  INT NOT NULL
  password_days  INT NOT NULL
  password_warning_days   INT NOT NULL) IN SYSTEM
```

IQ_SYSTEM_LOGIN_INFO_TABLE contains one row with the system default values for Sybase IQ Login Management. When a new user is added with sp_iqaddlogin, these default values are used for connection and password control.

user_admin_enabled Indicates whether Sybase IQ Login Management is enabled (Y) or disabled (N). Set by sp_iqmodifyadmin.

number_connects Maximum number of concurrent database connections by a single user. Default is 0: Sybase IQ does not enforce a maximum number of connections.

number_db_connects Number of connections allowed to the database.

password_days Number of days until a password expires. Default is 0: the password does not expire.

password_warning_days Number of days before a password expires at which Sybase IQ sends a warning to the user.

IQ_USER_LOGIN_INFO_TABLE system table

```
CREATE TABLE DBA.IQ_USER_LOGIN_INFO_TABLE
  userid   VARCHAR(128) NOT NULL UNIQUE,
  login_locked   CHAR(1) NOT NULL,
  number_connects   INT NOT NULL,
  password_created TIMESTAMP DEFAULT CURRENT TIMESTAMP
    NOT NULL
  password_days INT NOT NULL) IN SYSTEM
```

IQ_USER_LOGIN_INFO_TABLE contains Sybase IQ Login Management values for each user.

userid The user ID.

login_locked Indicates whether the user is locked (Y) or allowed to connect (N).

number_connects Maximum number of concurrent database connections. Default is 0: Sybase IQ does not enforce a maximum number of connections.

password_days Number of days until a password expires. Default is 0: the password does not expire.

password_warning_days Number of days before a password expires at which Sybase IQ sends a warning to the user.

SYSARTICLE system table

```
CREATE TABLE SYS.SYSARTICLE (
    publication_id UNSIGNED INT NOT NULL,
    table_id UNSIGNED INT NOT NULL,
    where_expr LONG VARCHAR,
    subscribe_by_expr LONG VARCHAR,
    query CHAR(1) NOT NULL,
    PRIMARY KEY ( publication_id, table_id ),
    FOREIGN KEY REFERENCES SYS.SYSPUBLICATION,
    FOREIGN KEY REFERENCES SYS.SYSTABLE
)
```

Each row of SYSARTICLE describes an article in a SQL Remote publication.

publication_id The publication of which this article is a part.

table_id Each article consists of columns and rows from a single table. This column contains the table ID for this table.

where_expr For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.

subscribe_by_expr For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.

SYSARTICLECOL system table

```
CREATE TABLE SYS.SYSARTICLECOL (
    publication_id UNSIGNED INT NOT NULL,
    table_id UNSIGNED INT NOT NULL,
    column_id UNSIGNED INT NOT NULL,
    PRIMARY KEY (publication_id, table_id, column_id),
    FOREIGN KEY REFERENCES SYS.SYSARTICLE,
    FOREIGN KEY REFERENCES SYS.SYSCOLUMN
)
```

Each row identifies a column in an article, identifying the column, the table it is in, and the publication it is part of.

publication_id A unique identifier for the publication of which the column is a part.

table_id The table to which the column belongs.

column_id The column identifier, from the SYSCOLUMN system table.

SYSCAPABILITY system table

```
CREATE TABLE SYS.SYSCAPABILITY (
    capid INT,
    capvalue CHAR(128),
    svrid INT,
    PRIMARY KEY (svrid),
    FOREIGN KEY REFERENCES SYS.SYSSERVERS,
    FOREIGN KEY REFERENCES SYS.SYSCAPABILITYNAME
)
```

Each row identifies a capability of a remote server.

capid An integer that identifies the capability, as listed in SYSCAPABILITYNAME.

capvalue The value of the capability.

svrid The server to which the capability applies, as listed in SYSSERVERS.

SYSCAPABILITYNAME system table

```
CREATE TABLE SYS.SYSCAPABILITYNAME (
    capid INT,
    capname CHAR(128),
    PRIMARY KEY (capid)
)
```

Each row identifies a capability.

capid An integer that identifies the capability.

capname The name of the capability.

SYSCHECK system table

```
CREATE TABLE SYS.SYSCHECK (
    check_id unsigned int NOT NULL,
    check_defn long varchar NOT NULL,
    primary key( check_id )
)
```

Each row identifies a named check constraint in a table.

check_id An identifier for the constraint.

check_defn The CHECK expression.

SYSCOLLATION system table

```
CREATE TABLE SYS.SYSCOLLATION (
    collation_id SMALLINT NOT NULL,
    collation_label CHAR(10) NOT NULL,
    collation_name CHAR(128) NOT NULL,
    collation_order BINARY(1280) NOT NULL,
    PRIMARY KEY ( collation_id )
)
```

This table contains the collation sequences available to Sybase IQ. You cannot modify the contents of this table.

collation_id A unique number identifying the collation sequence.

collation_label A string identifying each of the available collation sequences. The collation sequence to be used is selected when the database is created by specifying the collation label with the COLLATION option of the CREATE DATABASE command.

collation_name The name of the collation sequence.

collation_order An array of bytes defining how each of the 256 character codes are treated for comparison purposes. All string comparisons translate each character according to the collation order table before comparing the characters. For the different ASCII code pages, the only difference is how accented characters are sorted. In general, an accented character is sorted as if it were the same as the nonaccented character.

SYSCOLLATIONMAPPINGS system table

```
CREATE TABLE SYS.SYSCOLLATIONMAPPINGS (
    collation_label CHAR(10) NOT NULL,
    collation_name CHAR(128) NOT NULL,
    cs_label CHAR(128),
    so_case_label CHAR(128),
    so_caseless_label CHAR(128),
    jdk_label CHAR(128),
    PRIMARY KEY ( collation_label )
)
```

This table contains the collation mappings available in Sybase IQ. There is no way to modify the contents of this table.

collation_label A string identifying the collation sequence. The collation sequence to be used is selected when the database is created by specifying the collation label with the COLLATION option of the CREATE DATABASE command.

collation_name The collation name used to describe the character set encoding.

cs_label The GPG character set mapping label.

so_case_label The collation sort order for case-sensitive GPG character set mapping.

so_caseless_label The collation sort order for case-insensitive GPG character set mapping.

jdk_label The JDK character set label.

For newly-created databases, this table contains only one row with the database collation mapping. For databases created with version 12.5 or earlier, this table includes collation mappings for all built-in collations.

SYSCOLPERM system table

```
CREATE TABLE SYS.SYSCOLPERM (
    table_id UNSIGNED INT NOT NULL,
    grantee UNSIGNED INT NOT NULL,
    grantor UNSIGNED INT NOT NULL,
    column_id UNSIGNED INT NOT NULL,
    privilege_type SMALLINT NOT NULL,
    is_grantable CHAR( 1 ) NOT NULL,
    PRIMARY KEY ( table_id, grantee,
    grantor, column_id, privilege_type ),
    FOREIGN KEY grantee ( grantee ) REFERENCES
    SYS.SYSUSERPERM ( user_id ),
    FOREIGN KEY grantor ( grantor ) REFERENCES
    SYS.SYSUSERPERM ( user_id ),
    FOREIGN KEY REFERENCES SYS.SYSCOLUMN
)
```

The GRANT command can give UPDATE permission to individual columns in a table. Each column with UPDATE permission is recorded in one row of SYSCOLPERM.

table_id The table number for the table containing the column.

grantee The user number of the user ID given UPDATE permission on the column. If the grantee is the user number for the special PUBLIC user ID, the UPDATE permission is given to all user IDs.

grantor The user number of the user ID granting the permission.

column_id This column number, together with the *table_id*, identifies the column for which UPDATE permission has been granted.

privilege_type The number in this column indicates the kind of column permission (REFERENCES, SELECT or UPDATE).

is_grantable Indicates if the permission on the column was granted by the grantor to the grantee WITH GRANT OPTION. (Y/N).

SYSCOLUMN system table

```
CREATE TABLE SYS.SYSCOLUMN (
    table_id UNSIGNED INT NOT NULL,
    column_id UNSIGNED INT NOT NULL,
    pkey CHAR(1) NOT NULL,
    domain_id SMALLINT NOT NULL,
    nulls CHAR(1) NOT NULL,
    width SMALLINT NOT NULL,
    scale SMALLINT NOT NULL,
    estimate INT NOT NULL,
    max_identity BIGINT NOT NULL,
    column_name CHAR(128) NOT NULL,
    remarks LONG VARCHAR,
    "default" LONG VARCHAR,
    "check"LONG VARCHAR,
    user_type SMALLINT,
    format_str CHAR(128),
    column_type CHAR(1) NOT NULL,
    remote_name VARCHAR(128),
    remote_type UNSIGNED INT,
    PRIMARY KEY ( table_id, column_id ),
    FOREIGN KEY REFERENCES SYS.SYSTABLE,
    FOREIGN KEY REFERENCES SYS.SYSDOMAIN
    FOREIGN KEY REFERENCES SYS.SYSUSERTYPE
)
```

Each column in every table or view is described by one row in SYSCOLUMN.

table_id The table number uniquely identifies the table or view to which this column belongs.

column_id 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.

pkey Indicates whether this column is part of the primary key for the table (Y/N).

domain_id Identifies the data type for the column by the data type number listed in the SYSDOMAIN table.

nulls Indicates whether the NULL value is allowed in this column (Y/N).

width This column contains the length of string columns, the precision of numeric columns, 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 columns, and zero for all other data types.

estimate A self-tuning parameter for the optimizer. Sybase IQ "learns" from previous queries by adjusting guesses that are made by the optimizer.

max_identity The largest value of the column, if it is an AUTOINCREMENT, IDENTITY, or GLOBAL AUTOINCREMENT column. Sybase IQ does not support IDENTITY columns.

column_name The name of the column.

remarks A comment string.

"default" The default value for the column. This value is only used when an INSERT statement does not specify a value for the column.

"check" Any CHECK condition defined on the column.

Note The "default" value and "check" condition features are not currently supported by Sybase IQ.

---

user_type If the column is defined on a user-defined data type, the data type is held here.

format_str Currently unused.

column_type The type of column.

remote_name The name of the remote column.

remote_type The type of the remote column. This value is defined by the remote server or interface.

SYSCONSTRAINT system table

```
CREATE TABLE SYS.SYSCONSTRAINT (
    constraint_id unsigned int NOT NULL,
    table_id nsigned int NOT NULL,
    column_id unsigned int NULL,
    index_id unsigned int NULL,
    fkey_id mallint NULL,
    constraint_type char(1) NOT NULL,
    constraint_name char(128) NOT NULL,

    PRIMARY KEY( constraint_id ),
    UNIQUE( table_id, constraint_name )
)
```

Each row describes a named constraint.

constraint_id The unique constraint ID.

table_id The table ID of the table to which the constraint applies.

column_id The column ID of the column to which the constraint applies. The column is NULL for any constraints that are not column constraints.

index_id The index ID for a unique constraint. The column is NULL for all constraints that are not unique constraints.

fkey_id The foreign key ID for a foreign-key constraint. The column is NULL for all constraints that are not foreign-key constraints.

constraint_type Set to one of the following values:

- C is the constraint is a column check constraint.
- T if the constraint is a table constraint.
- P if the constraint is a primary key.
- F if the constraint is a foreign key.
- U if the constraint is a unique constraint.

constraint_name The name of the constraint.

SYSDOMAIN system table

```
CREATE TABLE SYS.SYSDOMAIN (
    domain_id SMALLINT NOT NULL,
    domain_name CHAR(128) NOT NULL,
    type_id SMALLINT,
    precision SMALLINT,
    PRIMARY KEY ( domain_id )
)
```

Each of the predefined data types (also called domains) in Sybase IQ is assigned a unique number. The SYSDOMAIN table is provided for informational purposes to show the association between these numbers and the appropriate data type. This table is never changed by Sybase IQ.

domain_id The unique number assigned to each data type. You cannot change these numbers.

domain_name A string containing the data type normally found in the CREATE TABLE command, such as char or integer.

type_id The ODBC data type. This corresponds to “data_type” in the Transact-SQL-compatible DBO.SYSTYPES table.

precision The number of significant digits that can be stored using this data type. The column value is NULL for nonnumeric data types.

SYSEVENT system table

```
CREATE TABLE SYS.SYSEVENT (
    event_id INT NOT NULL,
    creator UNSIGNED INT NOT NULL,
    event_name VARCHAR(128) NOT NULL UNIQUE,
    enabled CHAR(1) NOT NULL,
    location CHAR(1) NOT NULL,
    event_type_id INT NULL,
    action LONG VARCHAR NULL,
    external_action LONG VARCHAR NULL,
    condition LONG VARCHAR NULL,
    remarks LONG VARCHAR NULL,
    source LONG VARCHAR NULL,
    PRIMARY KEY ( event_id )
)
```

Each row in SYSEVENT describes an event created with CREATE EVENT.

event_id The unique number assigned to each event.

creator The user number of the owner of the event. The name of the user can be found by looking in SYSUSERPERM.

event_name The name of the event.

enabled (Y/N) Indicates whether or not the event is allowed to fire.

location The location where the event is allowed to fire:

- C = consolidated
- R = remote
- A = all

event_type_id For system events, the event type as listed in SYSEVENTTYPE.

action The event handler definition.

external_action Not used.

condition The WHERE condition used to control firing of the event handler.

remarks A comment string.

source This column contains the original source for the event handler if the PRESERVE_SOURCE_FORMAT option is ON. It is used to maintain the appearance of the original text. For more information, see "PRESERVE_SOURCE_FORMAT option [database]" in the *Sybase IQ System Administration Guide*.

SYSEVENTTYPE system table

```
CREATE TABLE SYS.SYSEVENTTYPE (
    event_type_id INT NOT NULL,
    name VARHCAR(128) NOT NULL UNIQUE,
    description LONG VARCHAR NULL,
    PRIMARY KEY ( event_type_id )
)
```

Each row in the SYSEVENTTYPE table describes a system event type which can be referenced by CREATE EVENT.

event_type_id The unique number assigned to each event type.

name The name of the system event type.

description A description of the system event type.

SYSEXTERNLOGINS system table

```
CREATE TABLE SYS.SYSEXTERNLOGINS (
    user_id UNSIGNED INT NOT NULL,
    srvid  INT NOT NULL,
    remote_login VARCHAR(128),
    remote_password VARBINARY(128),
    PRIMARY KEY ( user_id, srvid ),
FOREIGN KEY REFERENCES SYS.SYSUSERPERM,
FOREIGN KEY REFERENCES SYS.SYSSERVERS
)
```

Each row describes an external login for remote data access.

user_id The user ID on the local database.

srvid The remote server, as listed in SYSSERVERS.

remote_login The login name for this user, for the remote server.

remote_password The password for this user, for the remote server.

SYSFILE system table

```
CREATE TABLE SYS.SYSFILE (
    file_id SMALLINT NOT NULL,
    file_name LONG VARCHAR NOT NULL,
    dbspace_name CHAR(128) NOT NULL,
    store_type CHAR(8) NOT NULL,
    PRIMARY KEY ( file_id )
)
```

Every database consists of one or more operating system files. Each file is recorded in SYSFILE.

file_id Each file in a database is assigned a unique number. This file identifier is the primary key for SYSFILE. All system tables are stored in file_id 0.

file_name The database name is stored when a database is created. This name is for informational purposes only. For the SYSTEM dbspace, the file name always reflects the name when the data base was created. Changes to the file name are not reflected here.

dbspace_name Every file has a dbspace name that is unique. It is used in the CREATE TABLE command.

store_type Defines the file as belonging to either the Catalog Store (SA) or the IQ STORE.

SYSFKCOL system table

```
CREATE TABLE SYS.SYSFKCOL (
    foreign_table_id UNSIGNED INT NOT NULL,
    foreign_key_id SMALLINT NOT NULL,
    foreign_column_id UNSIGNED INT NOT NULL,
    primary_column_id UNSIGNED INT NOT NULL,
    PRIMARY KEY ( foreign_table_id,
    foreign_key_id, foreign_column_id ),
    FOREIGN KEY REFERENCES SYS.SYSFOREIGNKEY,
    FOREIGN KEY ( foreign_table_id,
    foreign_column_id ) REFERENCES
    SYS.SYSCOLUMN ( table_id, column_id )
)
```

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.

foreign_table_id The table number of the foreign table.

foreign_key_id The key number of the FOREIGN KEY for the foreign table. Together, *foreign_table_id* and *foreign_key_id* uniquely identify one row in SYSFOREIGNKEY, and the table number for the primary table can be obtained from that row.

foreign_column_id This column number, together with the *foreign_table_id*, identify the foreign column description in SYSCOLUMN.

primary_column_id This column number, together with the *primary_table_id* obtained from SYSFOREIGNKEY, identify the primary column description in SYSCOLUMN.

SYSFOREIGNKEY system table

```
CREATE TABLE SYS.SYSFOREIGNKEY (
    foreign_table_id UNSIGNED INT NOT NULL,
    foreign_key_id SMALLINT NOT NULL,
    primary_table_id UNSIGNED INT NOT NULL,
    root INT NOT NULL,
    check_on_commit CHAR(1) NOT NULL,
    nulls CHAR(1) NOT NULL,
    role CHAR(128) NOT NULL,
    remarks LONG VARCHAR,
    primary_index_id UNSIGNED INT NOT NULL,
    fk_not_enforced CHAR(1) NOT NULL
    hash_limit SMALLINTO NOT NULL,
    PRIMARY KEY ( foreign_table_id, foreign_key_id ),
    UNIQUE ( role, foreign_table_id ),
    FOREIGN KEY foreign_table ( foreign_table_id )
    REFERENCES SYS.SYSTABLE ( table_id ),
    FOREIGN KEY primary_table ( primary_table_id )
    REFERENCES SYS.SYSTABLE ( table_id )
)
```

A foreign key is a relationship between two tables—the foreign table and the primary table. Every foreign key is defined by one row in SYSFOREIGNKEY and one or more rows in SYSFKCOL. SYSFOREIGNKEY contains general information about the foreign key, while SYSFKCOL identifies the columns in the foreign key and associates each column in the foreign key with a column in the primary key of the primary table.

foreign_table_id The table number of the foreign table.

foreign_key_id Each foreign key has a foreign key number that is unique with respect to:

- The key number of all other foreign keys for the foreign table
- The key number of all foreign keys for the primary table
- The index number of all indexes for the foreign table

primary_table_id The table number of the primary table.

root Foreign keys are stored in the database as B-trees. The root identifies the location of the root of the B-tree in the database file.

check_on_commit Indicates whether INSERT and UPDATE commands should wait until the next COMMIT command to check if foreign keys are valid. A foreign key is valid if, for each row in the foreign table, the values in the columns of the foreign key either contain the NULL value or match the primary key values in some row of the primary table. (Y/N).

nulls Indicates whether the columns in the foreign key are allowed to contain the NULL value. This setting is independent of the nulls setting in the columns contained in the foreign key. (Y/N).

role The name of the relationship between the foreign table and the primary table. Unless otherwise specified, the role name is the same as the name of the primary table. The foreign table cannot have two foreign keys with the same role name.

remarks A comment string.

primary_index_id The *index_id* of the primary key, or root if the primary key is part of a combined index.

fk_not_enforced Is N if one of the tables is remote. (Y/N).

hash_limit Contains information about physical index representation.

SYSGROUP system table

```
CREATE TABLE SYS.SYSGROUP (
    group_id UNSIGNED INT NOT NULL,
    group_member UNSIGNED INT NOT NULL,
    PRIMARY KEY ( group_id, group_member ),
    FOREIGN KEY group_id ( group_id ) REFERENCES
    SYS.SYSUSERPERM ( user_id ),
    FOREIGN KEY group_member ( group_member )
    REFERENCES SYS.SYSUSERPERM ( user_id )
)
```

There is one row in SYSGROUP for every member of every group. This table describes a many-to-many relationship between groups and members. A group may have many members and a user may be a member of many groups.

group_id The user number of group.

group_member The user number of a member.

SYSINDEX system table

```
CREATE TABLE SYS.SYSINDEX (
    table_id UNSIGNED INT NOT NULL,
    index_id UNSIGNED INT NOT NULL,
    root INT NOT NULL,
    file_id SMALLINT NOT NULL,
    "unique" CHAR(1) NOT NULL,
    creator UNSIGNED INT NOT NULL,
    index_name CHAR(128) NOT NULL,
    remarks LONG VARCHAR,
    index_type CHAR(4) NOT NULL,
    index_owner CHAR(4) NOT NULL,
    hash_limit   SMALLINT NOT NULL,
    PRIMARY KEY ( table_id, index_id ),
    UNIQUE ( index_name, creator ),
    FOREIGN KEY REFERENCES SYS.SYSTABLE,
    FOREIGN KEY REFERENCES SYS.SYSFILE,
    FOREIGN KEY ( creator ) REFERENCES
    SYS.SYSUSERPERM ( user_id )
)
```

Each index in the database is described by one row in SYSINDEX.

table_id The table number uniquely identifies the table to which this index applies.

index_id Each index for one particular table is assigned a unique index number.

root Indexes are stored in the database as B-trees. The root identifies the location of the root of the B-tree in the database file.

file_id The index is completely contained in the file with this *file_id* (see “SYSFILE system table” on page 699).

“unique” Indicates whether the index is a unique index (“Y”), a nonunique index (“N”), or a unique constraint (“U”). A unique index prevents two rows in the indexed table from having the same values in the index columns.

creator The user number of the creator of the index.

index_name The name of the index. A user ID cannot have two indexes with the same name.

remarks A comment string.

index_type The type of index: FP (known as the default index), HG, HNG, LF, DATE, TIME, DTTM, CMP, WD, LD, or SA (for a non-IQ index created in the Catalog Store).

index_owner The name of the index owner: USER, IQ, SA, AUTO.

hash_limit For internal use.

SYSINFO system table

```
CREATE TABLE SYS.SYSINFO (
    page_size INTEGER NOT NULL,
    encryption CHAR(1) NOT NULL,
    blank_padding CHAR(1) NOT NULL,
    case_sensitivity CHAR(1) NOT NULL,
    default_collation CHAR(10) NOT NULL,
    database_version SMALLINT NOT NULL
    classes_version CHAR(10)
)
```

This table indicates the database characteristics as defined when the database was created using CREATE DATABASE. It always contains only one row.

page_size The Catalog page size specified to CREATE DATABASE. The default value is 1024.

encryption Whether encryption was specified with CREATE DATABASE. (Y/N).

blank_padding Whether the database was created to use blank padding for string comparisons in the database. (Y/N).

case_sensitivity Whether case sensitivity was specified with CREATE DATABASE. Case sensitivity affects value comparisons, but not table and column name comparisons. For example, if case sensitivity is enabled, the system catalog names such as SYSCATALOG must be specified in uppercase since that is how the name was spelled when it was created. (Y/N)

default_collation A string corresponding to the *collation_label* in SYSCOLLATE corresponding to the collation sequence specified with CREATE DATABASE. The collation sequence is used for all string comparisons, including searches for character strings as well as column and table name comparison.

database_version A small integer value indicating the database format. As newer versions of Sybase IQ become available, new features might require the format of the database file to change. The version number allows Sybase IQ software to determine if this database was created with a newer version of the software and thus cannot be understood by the software in use.

classes_version A small string describing the current version of the SYS.JAVA.CLASSES library that is currently installed on your computer.

SYSIQCOLUMN system table

```
CREATE TABLE SYS.SYSIQCOLUMN (
    table_id UNSIGNED INT NOT NULL,
    column_id UNSIGNED INT NOT NULL,
    link_table_id UNSIGNED INT NULL,
    link_column_id UNSIGNED INT NULL,
    max_length UNSIGNED INT NOT NULL,
    approx_unique_count ROWID
    cardinality ROWID NOT NULL,
    has_data CHAR(1) NOT NULL,
    has_original CHAR(1) NOT NULL,
    original_not_null CHAR(1) NOT NULL,
    original_unique CHAR(1) NOT NULL,
    info_location    HS_VDORECID NOT NULL,
    info_recid_size  UNSIGNED INT NOT NULL,
    info_location_size  UNSIGNED INT NOT NULL,
    PRIMARY KEY ( table_id, column_id )
)
```

Each column in every table is described by one row in SYSIQCOLUMN, which corresponds to a same row in SYSCOLUMN based on the primary key.

table_id The table number uniquely identifies the table to which this column belongs. It corresponds to the table_id column of SYSTABLE.

column_id 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 For internal use.

link_column_id For internal use.

max_length Indicates the maximum length allowed by the column.

approx_unique_count Approximate number of unique values (cardinality) of this column.

cardinality The actual number of unique values (cardinality) of this column.

has_data Indicates that the column contains data (T/F).

has_original Indicates the join index has the original data (T/F).

original_not_null Indicates the join index column with the original data was NOT NULL (T/F).

original_unique Indicates the join index column with the original data was UNIQUE (T/F).

info_location Not used. Always zero.

info_recid_size Not used. Always zero.

info_location_size Not used. Always zero.

SYSIQFILE system table

```
CREATE TABLE SYS.SYSIQFILE (
    file_id SMALLINT NOT NULL,
    start_block rowid NOT NULL,
    block_count rowid NOT NULL,
    create_time TIMESTAMP NOT NULL,
    segment_type CHAR(8) NOT NULL,
    allocated CHAR(1) NOT NULL,
    server_name  CHAR(30) NOT NULL,
    file_name    CHAR(128) NOT NULL,
    data_offset  UNSIGNED INT NOT NULL,
    PRIMARY KEY ( file_id )
    UNIQUE ( server_name, file_name )
)
```

Every database consists of one or more operating system files. Each dbspace and IQ Message file is recorded in SYSIQFILE (corresponding to a same entry in SYSFILE).

For multiplex, each server has unique entries in SYSIQFILE for its dbspace files. The rows in SYSIQFILE for a multiplex write server are maintained just like those for a nonmultiplex server. You must update the SYSIQFILE table on each server in a multiplex environment once it is modified (through CREATE or DROP DBSPACE commands) on any server, so that the tables contain identical information.

The initial CREATE or DROP DBSPACE must occur on the server that writes to the file (the specific server for IQ Temporary dbspaces, the write server for IQ Main dbspaces). That entry must then also be added to SYSIQFILE on each of the other servers. Sybase Central procedures that create and drop dbspaces, including the procedure that adds a new query server to the multiplex, maintain this agreement.

file_id Each file in a database is assigned a unique number. This file identifier is the primary key for SYSIQFILE, and it is linked to a same value in SYSFILE.

start_block Number of the first block.

block_count Number of blocks for this file (dbspace).

create_time Date and time the file was created.

segment_type Defines the type of segment: Main, Temp, or Msg.

allocated Defines whether the segment is preallocated (T) or autoallocated (F).

server_name For nonmultiplex databases and write servers, always blank. For multiplex query servers, always contains the query server's name.

file_name For nonmultiplex databases, always equal to SYS.SYSFILE *file_name* entry. For multiplex, the IQ dbspace name used by the multiplex server to open the IQ dbspace.

data_offset Used only for mixed-platform multiplex. Identifies the byte location of where the Sybase IQ data starts, relative to the beginning of the raw partition. Sybase IQ does not use the disk header block on a raw device. Because the disk header block is used by entities such as volume managers, Sybase IQ skips the first 65536 bytes of a raw device. Block numbers within Sybase IQ always start at 1. The first block would start at offset 65536.

SYSIQINDEX system table

```
CREATE TABLE SYS.SYSIQINDEX (
    table_id UNSIGNED INT NOT NULL,
    index_id UNSIGNED INT NOT NULL,
    max_key UNSIGNED INT NOT NULL,
    identity_location BINARY(16) NOT NULL,
    identity_size UNSIGNED INT NOT NULL,
    identity_location_size UNSIGNED INT NOT NULL,
    link_table_id UNSIGNED INT NOT NULL,
    link_index_id UNSIGNED INT NOT NULL,
    delimited by VARCHAR(1024),
    limit  UNSIGNED INT,
    PRIMARY KEY ( table_id, index_id )
)
```

Each index in the database is described by one row in SYSIQINDEX, which corresponds to an index in SYSINDEX.

table_id The table number uniquely identifies the table to which this index applies. It corresponds to the table_id column of SYSTABLE.

index_id Each index for one particular table is assigned a unique index number.

max_key For internal use.

identity_location For internal use.

identity_size For internal use.

identity_location_size For internal use.

link_table_id For internal use.

link_index_id For internal use.

delimited_by (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 (WD indexes only.) Maximum word length for WD index (between 1 and 255 bytes).

SYSIQINFO system table

```
CREATE TABLE SYS.SYSIQINFO (
    last_full_backup TIMESTAMP,
```

```
    last_incr_backup TIMESTAMP,
    create_time TIMESTAMP NOT NULL,
    update_time TIMESTAMP NOT NULL,
    file_format_version UNSIGNED INT NOT NULL,
    cat_format_version UNSIGNED INT NOT NULL
    sp_format_version UNSIGNED INT NOT NULL,
    block_size UNSIGNED INT NOT NULL
    chunk_size UNSIGNED INT NOT NULL,
    file_format_date CHAR(10) NOT NULL
    dbsig BINARY(136) NOT NULL
PRIMARY KEY ( create_time ),
)
```

This table indicates the database characteristics as defined when the Sybase IQ database was created using CREATE DATABASE. It always contains only one row.

last_full_backup Completion time of the most recent full backup.

last_incr_backup Completion time of the most recent incremental backup.

create_time Date and time created.

update_time Date and time of the last update.

file_format_version File format number of files for this database.

cat_format_version Catalog format number for this database.

sp_format_version Stored procedure format number for this database.

block_size Block size specified for the database.

chunk_size Number of blocks per chunk as determined by the block size and page size specified for the database.

file_format_date Date when file format number was last changed.

dbsig Used internally by catalog.

SYSIQJOININDEX system table

```
CREATE TABLE SYS.SYSIQJOININDEX (
    joinindex_id UNSIGNED INT NOT NULL,
    jvt_id UNSIGNED INT NOT NULL,
    joinindex_name CHAR(128) NOT NULL,
    joinindex_type CHAR(12) NOT NULL,
    creator UNSIGNED INT NOT NULL,
    join_info_location BINARY(16) NOT NULL,
    join_info_loc_size UNSIGNED INT NOT NULL,
    join_info_size UNSIGNED INT NOT NULL,
    block_map BINARY(32) NOT NULL,
    block_map_size UNSIGNED INT NOT NULL,
    vdo BINARY(256) NOT NULL,
    vdo_size UNSIGNED INT NOT NULL,
    commit_txn_id XACT_ID NOT NULL,
    txn_id XACT_ID NOT NULL,
    valid  CHAR(1) NOT NULL,
    remarks   LONG VARCHAR,
    PRIMARY KEY ( joinindex_id ),
    UNIQUE ( jvt_id, commit_txn_id, txn_id )
)
```

Each row of SYSIQJOININDEX describes one IQ join index in the database.

joinindex_id Each join index is assigned a unique number that is the primary key for SYSIQJOININDEX.

jvt_id For internal use.

joinindex_name Defines the name of the join index.

joinindex_type For internal use.

creator The number of the user that created the join index. The name of the user can be found by looking in SYSUSERPERM.

join_info_location For internal use.

join_info_loc_size For internal use.

join_info_size For internal use.

block_map For internal use.

block_map_size For internal use.

vdo For internal use.

vdo_size For internal use.

commit_txn_id For internal use.

txn_id For internal use.

valid Indicates whether this join index needs to be synchronized. Y indicates that it does not require synchronization, N indicates that it does require synchronization.

remarks A comment string.

SYSIQJOINIXCOLUMN system table

```
CREATE TABLE SYS.SYSIQJOINIXCOLUMN (
    joinindex_id UNSIGNED INT NOT NULL,
    left_table_id UNSIGNED INT NOT NULL,
    left_column_id UNSIGNED INT NOT NULL,
    join_type CHAR(4) NOT NULL,
    right_table_id UNSIGNED INT NOT NULL,
    right_column_id UNSIGNED INT NOT NULL,
    order_num    UNSIGNED INT NOT NULL,
    left_order_num  UNSIGNED INT NOT NULL,
    right_order_num  UNSIGNED INT NOT NULL,
    key_type  CHAR(8) NOT NULL,
    coalesce  CHAR(1)   NOT NULL,
    PRIMARY KEY ( joinindex_id, left_table_id,
left_column_id, right_table_id, right_column_id )
)
```

The rows of SYSIQJOINIXCOLUMN describe the columns that explicitly participate in a join index.

joinindex_id Corresponds to a join index value in SYSIQJOININDEX.

left_table_id Corresponds to a table value in SYSTABLE that forms the left side of the join operation.

left_column_id Corresponds to a column value in SYSCOLUMN that is part of the left side of the join.

join_type Only value currently supported is "=".

right_table_id Corresponds to a table value in SYSTABLE that forms the right side of the join operation.

right_column_id Corresponds to a column value in SYSCOLUMN that is part of the right side of the join.

order_num For internal use.

left_order_num For internal use.

right_order_num For internal use.

key_type 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 Not used.

SYSIQJOINIXTABLE system table

```
CREATE TABLE SYS.SYSIQJOINIXTABLE (
    table_id UNSIGNED INT NOT NULL,
    joinindex_id UNSIGNED INT NOT NULL,
    active UNSIGNED INT NOT NULL,
    PRIMARY KEY ( table_id, joinindex_id )
)
```

The rows of SYSIQJOINIXTABLE describe the tables that explicitly participate in a join index.

table_id Corresponds to a table value in SYSTABLE that is included in a join operation.

joinindex_id Corresponds to a join index value in SYSIQJOININDEX.

active Defines the number of times the table is used in the join index.

SYSIQTABLE system table

```
CREATE TABLE SYS.SYSIQTABLE (
    table_id UNSIGNED INT NOT NULL,
    block_map BINARY(32) NOT NULL,
    block_map_size UNSIGNED INT NOT NULL,
    vdo BINARY(256) NOT NULL,
    vdoid_size UNSIGNED INT NOT NULL,
    info_location    HS_VDORECID NOT NULL,
    info_recid_size  UNSIGNED INT NOT NULL,
    info_location_size  UNSIGNED INT NOT NULL,
    commit_txn_id UNSIGNED INT NOT NULL,
    txn_id UNSIGNED INT NOT NULL,
    join_id UNSIGNED INT NOT NULL,
```

```
    create_time TIMESTAMP NOT NULL,
    update_time TIMESTAMP NOT NULL,
    PRIMARY KEY ( table_id ),
    UNIQUE ( commit_txn_id, txn_id )
)
```

Each row of SYSIQTABLE describes one table in the database, which corresponds to a table entry in SYSTABLE.

table_id Each table is assigned a unique number (the table number) that is the primary key for SYSIQTABLE.

block_map For internal use.

block_map_size For internal use.

vdo For internal use.

vdoid_size For internal use.

info_location Not used. Always zero.

info_recid_size Not used. Always zero.

info_location_size Not used. Always zero.

commit_txn_id For internal use.

txn_id For internal use.

join_id For internal use.

create_time Date and time the IQ table was created.

update_time Last date and time the IQ table was modified.

SYSIXCOL system table

```
CREATE TABLE SYS.SYSIXCOL (
    table_id UNSIGNED INT NOT NULL,
    index_id UNSIGNED INT NOT NULL,
    sequence SMALLINT NOT NULL,
    column_id UNSIGNED INT NOT NULL,
    "order" CHAR(1) NOT NULL,
    PRIMARY KEY ( table_id, index_id, sequence ),
    FOREIGN KEY REFERENCES SYS.SYSINDEX,
    FOREIGN KEY REFERENCES SYS.SYSCOLUMN
)
```

Every index has one row in SYSIXCOL for each column in the index.

table_id Identifies the table to which the index applies.

index_id Identifies in which index this column is used. Together, *table_id* and *index_id* identify one index described in SYSINDEX.

sequence Each column in an index is assigned a unique number starting at 0. The order of these numbers determines the relative significance of the columns in the index. The most important column has sequence number 0.

column_id The column number identifies which column is indexed. Together, *table_id* and *column_id* identify one column in SYSCOLUMN.

"order" Indicates whether this column in the index is kept in ascending or descending order ("A" or "D").

SYSJAR system table

```
CREATE TABLE SYS.SYSJAR(
    jar_id INTEGER NOT NULL,
    creator UNSIGNED INT NOT NULL,
    jar_name LONG VARCHAR NOT NULL,
    jar_file LONG VARCHAR,
    create_time TIMESTAMP NOT NULL,
    update_time TIMESTAMP NOT NULL,
    remarks LONG VARCHAR,
    PRIMARY KEY ( jar_id )
)
```

jar_id A field containing the ID of the JAR file.

creator This user number identifies the creator of the JAR file. The name of the user can be found by looking in SYSUSERPERM.

jar_name Name of the JAR file.

jar_file File name of the JAR file.

create_time Time the JAR file was created.

update_time Time the JAR file was last updated.

remarks A comment string.

SYSJARCOMPONENT system table

```
CREATE TABLE SYS.SYSJARCOMPONENT(
    component_id INT NOT NULL,
    jar_id INT,
    component_name LONG VARCHAR,
    component_type CHAR(1),
    create_time TIMESTAMP NOT NULL,
    contents LONG BINARY,
    remarks LONG VARCHAR,
    PRIMARY KEY ( component_id ),
    FOREIGN KEY REFERENCES SYS.SYSJAR
)
```

component_id Primary key containing the ID of the component.

jar_id Field containing the ID of the JAR file. This field also references the SYSJAR system table.

component_name Name of the component.

component_type Type of the component.

create_time Field containing the creation time of the component.

contents Byte code of the JAR file.

remarks A comment string.

SYSJAVACLASS system table

```
CREATE TABLE SYS.SYSJAVACLASS(
    class_id INT NOT NULL,
    replaced_by INT,
    creator UNSIGNED INT NOT NULL,
    jar_id INT,
    type_id UNSIGNED INT,
    class_name LONG VARCHAR NOT NULL,
    public CHAR(1) NOT NULL,
    component_id INT,
    create_time TIMESTAMP NOT NULL,
    update_time TIMESTAMP NOT NULL,
    class_descriptor LONG BINARY,
    remarks LONG VARCHAR,
    PRIMARY KEY ( class_id ),
    FOREIGN KEY ( replaced_by ) REFERENCES
```

```
o.SYSJAVACLASS ( class_id ),
    FOREIGN KEY ( creator ) REFERENCES
SYS.SYSUSERPERM ( user_id ),
    FOREIGN KEY REFERENCES SYS.SYSUSERTYPE
    FOREIGN KEY REFERENCES SYS.SYSJARCOMPONENT
)
```

The SYSJAVACLASS system table contains all information related to Java classes.

class_id ID of the Java class.

replaced_by References the primary key field class_id.

creator user_id of the creator of the class. This field references the user_id field in the SYSUSERPERM system table to obtain the name of the user.

jar_id ID of the JAR file from which the class came.

type_id ID of the user type. The field references the SYSUSERTYPE system table to obtain the ID of the user.

class_name Name of the Java class.

public Whether the class is public or private.

component_id References the SYSJARCOMPONENT system table and contains the ID of the component.

create_time Creation time of the component.

update_time Last update time of the component.

class_descriptor Byte code of the JAR file.

remarks A comment string.

SYSLOGIN system table

```
CREATE TABLE SYS.SYSLOGIN (
    integrated_login_id CHAR(128) NOT NULL,
    login_uid UNSIGNED INT NOT NULL,
    remarks LONG VARCHAR,
    PRIMARY KEY ( integrated_login_id )
)
```

This table contains all the user profile names that can be used to connect to the database using an integrated login. As a security measure, only users with DBA authority can view the contents of this table.

integrated_login_id A string value containing the user profile name used to map to a user ID in the database. When a user successfully logs on using this user profile name, and the database is enabled to accept integrated logons, the user can connect to the database without providing a user ID or password.

login_uid A foreign key to the system table SYSUSERPERM.

remarks A comment string.

SYSOPTION system table

```
CREATE TABLE SYS.SYSOPTION (
    user_id UNSIGNED INT NOT NULL,
    "option" CHAR(128) NOT NULL,
    "setting" LONG VARCHAR NOT NULL,
    PRIMARY KEY ( user_id, "option" ),
    FOREIGN KEY REFERENCES SYS.SYSUSERPERM
)
```

Options settings are stored in the SYSOPTION table by the SET command. Each user can have their own setting for each option. In addition, settings for the PUBLIC user ID define the default settings to be used for user IDs that do not have their own setting.

Note If you query the option column of this table in a case-sensitive database, you must match case of the option. For example, the MAIN_CACHE_MEMORY_MB option is stored in SYSOPTION as Main_Cache_Memory_MB. You can run a select * from the SYSOPTION table to see the exact case of the option.

---

user_id User number to whom this option setting applies.

"option" Name of the option.

"setting" Current setting for the named option.

If you incorrectly type the name of an option when you are setting it, the incorrect name is saved in the SYSOPTION table. You can remove the incorrectly typed name from the SYSOPTION table by setting the option PUBLIC with an equality after the option name and no value:

```
SET OPTION PUBLIC.a_mistyped_name=;
```

SYSOPTIONDEFAULTS system table

```
create table DBA.SYSOPTIONDEFAULTS (
    option_name varchar(128),
    default_value varchar(40)
)
```

The SYSOPTIONDEFAULTS table stores the default option settings. These values do not change. The sp_iqcheckoptions stored procedure compares the default value in the SYSOPTIONDEFAULTS table to the current setting of the option in the SYSOPTION table and displays the values that have changed for the connected user.

Note If you query the option_name column of this table in a case-sensitive database, you must match case of the option. For example, the MAIN_CACHE_MEMORY_MB option is stored in SYSOPTIONDEFAULTS as Main_Cache_Memory_MB. You can run a `select *` from the SYSOPTIONDEFAULTS table to see the exact case of the option.

---

option_name Name of the option.

default_value Default value of the option.

SYSPROCEDURE system table

```
CREATE TABLE SYS.SYSPROCEDURE (
    proc_id UNSIGNED INT NOT NULL,
    creator UNSIGNED INT NOT NULL,
    proc_name CHAR(128) NOT NULL,
    proc_defn LONG VARCHAR,
    remarks LONG VARCHAR,
    replicate CHAR(1) NOT NULL,
```

```
    srvid INT NOT NULL,
    source LONG VARCHAR,
    avg_num_rows FLOAT,
    avg_costs FLOAT,
    stats LONG BINARY,
    PRIMARY KEY ( proc_id ),
    UNIQUE ( proc_name, creator ),
    FOREIGN KEY ( creator ) REFERENCES
       SYS.SYSUSERPERM ( user_id )
)
```

Each procedure in the database is described by one row in SYSPROCEDURE.

proc_id Each procedure is assigned a unique number (the procedure number) that is the primary key for SYSPROCEDURE.

creator This user number identifies the owner of the procedure. The name of the user can be found by looking in SYSUSERPERM.

proc_name Name of the procedure. One creator cannot have two procedures with the same name.

proc_defn Command used to create the procedure.

remarks A comment string.

replicate Holds a Y if the procedure is a primary data source in a Replication Server installation, and an N if not.

srvid If a procedure on a remote database server, indicates the remote server.

source Contains the original source for the procedure if the PRESERVE_SOURCE_FORMAT option is ON. It is used to maintain the appearance of the original text. For more information, see "PRESERVE_SOURCE_FORMAT option [database]" on page 137.

avg_num_rows Information collected for use in query optimization when the procedure appears in the FROM clause.

avg_cost Information collected for use in query optimization when the procedure appears in the FROM clause.

stats Information collected for use in query optimization when the procedure appears in the FROM clause.

SYSPROCPARM system table

```
CREATE TABLE SYS.SYSPROCPARM (
    proc_id UNSIGNED INT NOT NULL,
    parm_id SMALLINT NOT NULL,
    parm_type SMALLINT NOT NULL,
    parm_mode_in CHAR(1) NOT NULL,
    parm_mode_out CHAR(1) NOT NULL,
    domain_id SMALLINT NOT NULL,
    width SMALLINT NOT NULL,
    scale SMALLINT NOT NULL,
    parm_name CHAR(128) NOT NULL,
    remarks LONG VARCHAR,
    "default" LONG VARCHAR,
    user_type INTEGER,
    PRIMARY KEY ( proc_id, parm_id ),
    FOREIGN KEY REFERENCES SYS.SYSPROCEDURE,
    FOREIGN KEY REFERENCES SYS.SYSDOMAIN
)
```

Each parameter to a procedure in the database is described by one row in SYSPROCPARM.

proc_id Procedure number that uniquely identifies the procedure to which this parameter belongs.

parm_id Each procedure starts numbering parameters at 1. The order of parameter numbers corresponds to the order in which they were defined.

parm_type The type of parameter is one of the following:

- Normal parameter (variable)
- Result variable – used with procedure that return result sets
- SQLSTATE error value
- SQLCODE error value

parm_mode_in (Y/N) Indicates whether this parameter supplies a value to the procedure (IN or INOUT parameters).

parm_mode_out Indicates whether this parameter returns a value from the procedure (OUT or INOUT parameters). (Y/N).

domain_id Identifies the data type for the parameter by the data type number listed in the SYSDOMAIN table.

width Length of string parameters, precision of numeric parameters, and number of bytes of storage for all other data types.

scale Number of digits after the decimal point for numeric data type parameters, and zero for all other data types.

parm_name Name of the parameter.

remarks A comment string.

default Default value for the parameter, held as a string.

user_type User type of the parameter.

SYSPROCPERM system table

```
CREATE TABLE SYS.SYSPROCPERM (
    proc_id UNSIGNED INT NOT NULL,
    grantee UNSIGNED INT NOT NULL,
    PRIMARY KEY ( proc_id, grantee )
    FOREIGN KEY ( grantee ) REFERENCES
    SYS.SYSUSERPERM ( user_id ),
    FOREIGN KEY REFERENCES SYS.SYSPROCEDURE
)
```

Only users who have been granted permission can call a procedure. Each row of the SYSPROCPERM table corresponds to one user granted permission to call one procedure.

proc_id The procedure number uniquely identifies the procedure for which permission has been granted.

grantee The user number of the user ID receiving the permission.

SYSPUBLICATION system table

```
CREATE TABLE SYS.SYSPUBLICATION (
    publication_id UNSIGNED INT NOT NULL,
    creator UNSIGNED INT NOT NULL,
    publication_name CHAR(128) NOT NULL,
    remarks LONG VARCHAR,
    PRIMARY KEY ( publication_id ),
    FOREIGN KEY ( creator )
    REFERENCES SYS.SYSUSERPERM (user_id )
)
```

Each row describes a SQL Remote publication.

publication_id Unique identifying number for the publication.

creator Owner of the publication.

publication_name Name of the publication, which must be a valid identifier.

remarks Descriptive comments.

SYSREMOTEOPTION system table

```
CREATE table SYS.SYSREMOTEOPTION (
    option_id UNSIGNED INT NOT NULL,
    user_id UNSIGNED INT NOT NULL,
    "setting" VARCHAR( 255 ) NOT NULL,
    PRIMARY KEY ( option_id, user_id ),
    FOREIGN KEY REFERENCES SYS.SYSREMOTEOPTIONTYPE
)
```

Each row describes the values of a SQL Remote message link parameter.

option_id Identification number for the message link parameter.

user_id User ID for which the parameter is set.

"setting" Value of the message link parameter.

SYSREMOTEOPTIONTYPE system table

```
CREATE table SYS.SYSREMOTEOPTIONTYPE (
    option_id  UNSIGNED INT NOT NULL,
    type_id SMALLINT NOT NULL,
    "option" VARCHAR( 128 )  NOT NULL,
    PRIMARY KEY ( option_id ),
    FOREIGN KEY REFERENCES SYS.SYSREMOTETYPE
)
```

Each row describes one of the SQL Remote message link parameters.

option_id Identification number for the message link parameter.

type_id Identification number for the message type that uses this parameter.

"option" Name of the message link parameter.

SYSREMOTETYPE system table

```
CREATE TABLE SYS.SYSREMOTETYPE (
    type_id SMALLINT NOT NULL,
    type_name CHAR(128) NOT NULL,
    publisher_address LONG VARCHAR NOT NULL,
    remarks LONG VARCHAR,
    PRIMARY KEY ( type_id )
)
```

The SYSREMOTETYPE system table contains information about SQL Remote.

type_id Identifies which of the message systems supported by SQL Remote is used to send messages to this user.

type_name Name of the message system supported by SQL Remote.

publisher_address Address of the remote database publisher.

remarks Descriptive comments.

SYSREMOTEUSER system table

```
CREATE TABLE SYS.SYSREMOTEUSER (
    user_id UNSIGNED INT NOT NULL,
    consolidate CHAR(1) NOT NULL,
    type_id SMALLINT NOT NULL,
    address LONG VARCHAR NOT NULL,
    frequency CHAR(1) NOT NULL,
    send_time TIME,
    log_send NUMERIC(20,0) NOT NULL,
    time_sent TIMESTAMP,
    log_sent NUMERIC(20,0) NOT NULL,
    confirm_sent NUMERIC(20,0) NOT NULL,
    send_count INTEGER NOT NULL,
    resend_count INTEGER NOT NULL,
    time_received TIMESTAMP,
    log_received NUMERIC(20,0) NOT NULL,
    confirm_received NUMERIC(20,0),
```

```
    receive_count INTEGER NOT NULL,
    rereceive_count INTEGER NOT NULL,
    PRIMARY KEY (user_id),
    FOREIGN KEY REFERENCES SYS.SYSUSERPERM
)
```

Each row describes a user ID with REMOTE permissions (a subscriber), together with the status of SQL Remote messages sent to and from that user.

user_id ID of the user with REMOTE permissions.

consolidate Contains either an N to indicate a user granted REMOTE permissions, or a Y to indicate a user granted CONSOLIDATE permissions.

type_id Identifies which of the of the message systems supported by SQL Remote is to be used to send messages to this user.

address The address to which SQL Remote messages are to be sent. The address must be appropriate for the *address_type*.

frequency How frequently SQL Remote messages are to be sent.

send_time The next time messages are to be sent to this user.

log_send Messages are sent only to subscribers for whom *log_send* is greater than *log_sent*.

time_sent Time the most recent message was sent to this subscriber.

log_sent Log offset for the most recently sent operation.

confirm_sent Log offset for the most recently confirmed operation from this subscriber.

send_count Number of SQL Remote messages that have been sent.

resend_count Counter to ensure messages are applied only once at the subscriber database.

time_received Time the most recent message was received from this subscriber.

log_received Log offset in the subscriber’s database for the operation most recently received at the current database.

confirm_received The log offset in the subscriber's database for the most recent operation for which a confirmation message has been sent.

receive_count Number of messages received.

rereceive_count Counter to ensure messages are applied only once at the current database.

SYSSCHEDULE system table

```
CREATE TABLE SYS.SYSSCHEDULE (
    event_id INT NOT NULL,
    sched_name VARCHAR(128) NOT NULL,
    recurring TINYINT NOT NULL,
    start_time TIME NOT NULL,
    stop_time TIME NULL,
    start_date DATE NULL,
    days_of_week TINYINT NULL,
    days_of_month UNSIGNED INT NULL,
    interval_units CHAR(10) NULL,
    interval_amt INT NULL,
    PRIMARY KEY ( event_id, sched_name )
)
```

Each row in SYSSCHEDULE describes the times at which an event is to fire, as specified by the SCHEDULE clause of CREATE EVENT.

event_id Unique number assigned to each event.

sched_name Name associated with a schedule.

recurring (0/1) Indicates whether the schedule is repeating.

start_time Schedule start time.

stop_time Schedule stop time, if BETWEEN was used.

start_date First date on which the event is scheduled to execute.

days_of_week Bit mask indicating the days of the week on which the event is scheduled:

- x01 – Sunday.
- x02 – Monday.
- x04 – Tuesday.
- x08 – Wednesday.
- x10 – Thursday.
- x20 – Friday.
- x40 – Saturday.

days_of_month A bit mask indicating the days of the month on which the event is scheduled:

- x01 – first day of the month.

- x02 – second day of the month.
- x40000000 – 31st day of the month.
- x80000000 – last day of the month.

interval_units The interval unit specified by EVERY:

- HH – hours.
- NN – minutes.
- SS – seconds.

interval_amt The period specified by EVERY.

SYSSERVERS system table

```
CREATE TABLE SYS.SYSSERVERS (
    srvid  INT  NOT NULL,
    srvname VARCHAR(128) NOT NULL,
    srvclass LONG VARCHAR NOT NULL,
    srvinfo LONG VARCHAR,
    srvreadonly CHAR(1) NOT NULL,
    PRIMARY KEY ( srvid )
)
```

Each row describes a remote server.

srvid Identifier for the remote server.

srvname Name of the remote server.

srvclass Server class, as specified in the CREATE SERVER statement.

srvinfo Server information.

srvreadonly Y if the server is read only, and N otherwise.

SYSSQLSERVERTYPE system table

```
CREATE TABLE SYS.SYSSQLSERVERTYPE (
    ss_user_type SMALLINT NOT NULL,
    ss_domain_id SMALLINT NOT NULL,
    ss_type_name VARCHAR(30) NOT NULL,
    primary_sa_domain_id SMALLINT NOT NULL,
    primary_sa_user_type SMALLINT NULL,
    PRIMARY KEY ( type_id )
)
```

This table contains information relating to compatibility with Adaptive Server Enterprise.

ss_user_type Adaptive Server Enterprise user type.

ss_domain_id Adaptive Server Enterprise domain ID.

ss_type_name Adaptive Server Enterprise type name.

primary_sa_domain_id Primary domain ID.

primary_sa_user_type Primary user type.

SYSSUBSCRIPTION system table

```
CREATE TABLE SYS.SYSSUBSCRIPTION (
    publication_id UNSIGNED INT NOT NULL,
    user_id UNSIGNED INT NOT NULL,
    subscribe_by CHAR(128) NOT NULL,
    created NUMERIC(20,0) NOT NULL,
    started NUMERIC(20,0),
    PRIMARY KEY (publication_id, user_id,
subscribe_by),
    FOREIGN KEY REFERENCES SYS.SYSPUBLICATION,
    FOREIGN KEY REFERENCES SYS.SYSREMOTEUSER
);
```

Each row describes a subscription from one user ID (which must have REMOTE permissions) to one publication.

publication_id Identifier for the publication to which the user ID is subscribed.

user_id User ID that is subscribed to the publication.

subscribe_by For publications with a SUBSCRIBE BY expression, holds the matching value for this subscription.

created Offset in the transaction log at which the subscription was created.

started Offset in the transaction log at which the subscription was started.

SYSTABLE system table

```
CREATE TABLE SYS.SYSTABLE (
    table_id UNSIGNED INT NOT NULL,
    file_id SMALLINT NOT NULL,
    count UNSIGNED BIGINT NOT NULL,
    first_page INT NOT NULL,
    last_page INT NOT NULL,
    primary_root INT NOT NULL,
    creator UNSIGNED INT NOT NULL,
    first_ext_page   INT NOT NULL,
    last_ext_page    INT NOT NULL,
    table_page_count    INT NOT NULL,
    ext_page_count   INT NOT NULL,
    table_name CHAR(128) NOT NULL,
    table_type CHAR(10) NOT NULL,
    view_def LONG VARCHAR,
    remarks LONG VARCHAR,
    replicate CHAR(1) NOT NULL,
    "existing_obj" CHAR(1),
    remote_location LONG VARCHAR,
    remote_objtype CHAR(1),
    srvid INTEGER,
    server_type CHAR(4) NOT NULL,
    primary_hash_limit  SMALLINT NOT NULL,
    PRIMARY KEY ( table_id ),
    UNIQUE ( table_name, creator ),
    FOREIGN KEY ( creator ) REFERENCES
    SYS.SYSUSERPERM ( user_id ),
    FOREIGN KEY REFERENCES SYS.SYSFILE
)
```

Each row of SYSTABLE describes one table or view in the database.

table_id Each table or view is assigned a unique number (the table number) that is the primary key for SYSTABLE.

file_id The file number indicates which database file contains the table. The *file_id* is a FOREIGN KEY for SYSFILE.

count The count is always 0 for a view or an IQ table.

first_page Each Sybase IQ database is divided into a number of fixed size pages. This value identifies the first page containing information for this table, and is used internally to find the start of this table. The *first_page* is always 0 for a view.

last_page Last page containing information for this table. The *last_page* is always 0 for a view.

primary_root Primary keys are stored in the database as B-trees. The *primary_root* locates the root of the B-tree for the primary key for the table. It is 0 for a view and for a table with no primary key.

creator This user number identifies the owner of the table or view. The name of the user can be found by looking in SYSUSERPERM.

table_name Name of the table or view. One creator cannot have two tables or views with the same name.

first_ext_page For internal use.

last_ext_page For internal use.

table_page_count For internal use.

ext_page_count For internal use.

table_type This column is BASE for base tables and VIEW for views. It is GBL TEMP for global temporary tables and JVT for join indexes. No entry is created for local temporary tables.

view_def For a view, this column contains the CREATE VIEW command used to create the view. For a table, this column contains any CHECK constraints for the table.

remarks A comment string.

replicate Holds a Y if the table is a primary data source in a Replication Server installation, or an N if not.

“existing_obj” Indicates whether the table previously existed or not. (Y/N).

remote_location Indicates the storage location of the remote object.

remote_objtype Indicates the type of remote object: 'T' if table; 'V' if view; 'R' if RPC; 'B' if JavaBean.

srvid The unique ID for the server.

server_type Indicates whether the table was created in the Catalog Store (SA) or IQ Store.

primary_hash_limit For internal use.

SYSTABLEPERM system table

```
CREATE TABLE SYS.SYSTABLEPERM (
    stable_id UNSIGNED INT NOT NULL,
    grantee UNSIGNED INT NOT NULL,
    grantor UNSIGNED INT NOT NULL,
    ttable_id UNSIGNED INT NOT NULL,
    selectauth CHAR(1) NOT NULL,
    insertauth CHAR(1) NOT NULL,
    deleteauth CHAR(1) NOT NULL,
    updateauth CHAR(1) NOT NULL,
    updatecols CHAR(1) NOT NULL,
    alterauth CHAR(1) NOT NULL,
    referenceauth CHAR(1) NOT NULL,
    PRIMARY KEY ( stable_id, grantee, grantor ),
    FOREIGN KEY ( stable_id )
    REFERENCES SYS.SYSTABLE ( table_id ),
    FOREIGN KEY future ( ttable_id )
    REFERENCES SYS.SYSTABLE ( table_id ),
    FOREIGN KEY grantee ( grantee ) REFERENCES
    SYS.SYSUSERPERM ( user_id ),
    FOREIGN KEY grantor ( grantor )
    REFERENCES SYS.SYSUSERPERM ( user_id )
)
```

Permissions given by the GRANT command are stored in SYSTABLEPERM. Each row in this table corresponds to one table, one user ID granting the permission (grantor) and one user ID granted the permission (grantee).

There are several types of permission that can be granted. Each permission can have one of the following three values.

- N – no, the grantee has not been granted this permission by the grantor.
- Y – yes, the grantee has been given this permission by the grantor.

- G – the grantee has been given this permission. In addition, the grantee can grant the same permission to another user.

Note The grantee might have been given permission for the same table by another grantor. If so, this information is recorded in a different row of SYSTABLEPERM.

---

stable_id Table number of the table or view to which the permissions apply.

grantor User number of the user ID granting the permission.

grantee Tser number of the user ID receiving the permission.

ttable_id In the current version of Sybase IQ, this table number is always the same as *stable_id*.

selectauth Indicates whether SELECT permission has been granted. (Y/N/G).

insertauth Indicates whether INSERT permission has been granted. (Y/N/G) .

deleteauth Indicates whether DELETE permission has been granted. (Y/N/G).

updateauth Indicates whether UPDATE permission has been granted for all columns in the table. (Only UPDATE permission can be given on individual columns. All other permissions are for all columns in a table.) (Y/N/G).

updatecols (Y/N) Indicates whether UPDATE permission has only been granted for some of the columns in the table. If updatecols has the value Y, there is one or more rows in SYSCOLPERM granting update permission for the columns in this table.

alterauth (Y/N/G) Indicates whether ALTER permission has been granted.

referenceauth (Y/N/G) Indicates whether REFERENCE permission has been granted.

SYSTYPEMAP system table

```
CREATE TABLE SYS.SYSTYPEMAP (
    ss_user_type SMALLINT NOT NULL,
    sa_domain_id SMALLINT NOT NULL,
    sa_user_type SMALLINT NULL,
    nullable CHAR(1) NULL,
    FOREIGN KEY REFERENCES SYS.SYSSQLSERVERTYPE
)
```

The SYSTYPEMAP system table contains the compatibility mapping values for the SYSSQLSERVERTYPE system table.

ss_user_type Adaptive Server Enterprise user type.

sa_domain_id Domain ID.

sa_user_type User type.

nullable Whether or not the type can be NULL.

primary_sa_user_type Primary user type.

SYSUSERMESSAGES system table

```
CREATE TABLE SYS.SYSUSERMESSAGES (
    error INT NOT NULL,
    uid UNSIGNED INT NOT NULL,
    description VARCHAR(255) NOT NULL,
    langid SMALLINT NOT NULL,
    UNIQUE ( error, langid )
)
```

Each row holds a user-defined message for an error condition.

error Unique identifying number for the error condition.

uid User ID defining the message.

description Message corresponding to the error condition.

langid Reserved.

SYSUSERPERM system table

```
CREATE TABLE SYS.SYSUSERPERM (
    user_id UNSIGNED INT NOT NULL,
    user_name CHAR(128) NOT NULL UNIQUE,
    password BINARY(36),
    resourceauth CHAR(1) NOT NULL,
    dbaauth CHAR(1) NOT NULL,
    scheduleauth CHAR(1) NOT NULL,
    publishauth CHAR(1) NOT NULL,
    remotedbaauth CHAR(1) NOT NULL,
    user_group CHAR(1) NOT NULL,
    remarks LONG VARCHAR,
    PRIMARY KEY ( user_id )
)
```

Note SYSUSERPERM contains passwords requires DBA permissions to SELECT from the table.

Each row of SYSUSERPERM describes one user ID.

user_id Each new user ID is assigned a unique number (the user number) that is the primary key for SYSUSERPERM.

user_name String containing the name for the user ID. Each userid must have a unique name.

password Password for the user ID. The password contains the NULL value for the special user IDs SYS and PUBLIC, preventing anyone from connecting to these user IDs.

resourceauth Indicates whether the user has RESOURCE authority. Resource authority is required to create tables. (Y/N).

dbaauth Indicates whether the user has database administrator (DBA) authority. DBA authority is very powerful, and should be restricted to as few user IDs as possible for security purposes. (Y/N).

scheduleauth Indicates whether the user has SCHEDULE authority. This is currently not used by Sybase IQ. (Y/N).

publishauth Indicates whether the user has the SQL Remote publisher authority. (Y/N).

remotedbaauth Indicates whether the user has the SQL Remote remote DBA authority. (Y/N).

user_group Indicates whether the user is a group. (Y/N).

remarks Comment string.

When a database is initialized, the following user IDs are created:

- SYS – Creator of all the system tables.
- PUBLIC – Special user ID used to record PUBLIC permissions.
- DBA – The database administrator user ID is the only usable user ID in an initialized system. The initial password is SQL.

There is no way to connect to the SYS or PUBLIC user IDs.

SYSUSERTYPE system table

```
CREATE TABLE SYS.SYSUSERTYPE (
    type_id SMALLINT NOT NULL,
    creator UNSIGNED INT NOT NULL,
    domain_id UNSIGNED INT NOT NULL,
    nulls CHAR(1) NOT NULL,
    width SMALLINT NOT NULL,
    scale SMALLINT NOT NULL,
    type_name CHAR(128) NOT NULL,
    "default" LONG VARCHAR NULL,
    "check" LONG VARCHAR NULL,
    format_str CHAR(128),
    super_type_id SMALLINT NULL,
    UNIQUE ( type_name ),
    PRIMARY KEY ( type_id ),
    FOREIGN KEY ( creator )
    REFERENCES SYS.SYSUSERPERM ( user_id ),
    FOREIGN KEY REFERENCES SYS.SYSDOMAIN,
    FOREIGN KEY ( super_type_id )
    REFERENCES SYS.SYSUSERTYPE ( type_id )
)
```

Each row holds a description of a user-defined data type.

type_id Unique identifying number for the user-defined data type.

creator Owner of the data type.

domain_id Identifies the data type for the column by the data type number listed in the SYSDOMAIN table.

nulls Y indicates that the user-defined data type does allow nulls. N indicates that the data type does not allow nulls.

width Length of string columns, the precision of numeric columns, and the number of bytes of storage for all other data types.

scale Number of digits after the decimal point for numeric data type columns, and zero for all other data types.

type_name Name for the data type, which must be a valid identifier.

"default" Default value for the data type.

"check" CHECK condition for the data type.

Note The "default" value and "check" condition features are not currently supported by Sybase IQ.

---

format_str Currently unused.

SYSWEBSERVICE system table

```
CREATE TABLE SYS.SYSWEBSERVICE (
    service_id UNSIGNED INT NOT NULL,
    service_name CHAR(128) NOT NULL,
    service_type VARCHAR(40) NOT NULL,
    auth_required CHAR(1) NOT NULL,
    secure_required CHAR(1) NOT NULL,
    url_path CHAR(1) NOT NULL,
    user_id UNSIGNED INT,
    parameter VARCHAR(250)
    statement LONG VARCHAR,
    remarks LONG VARCHAR,
    super_type_id SMALLINT NULL,
    UNIQUE ( type_name ),
    PRIMARY KEY ( service_id )
)
)
```

Each row holds a description of a web service.

service_id A unique identifying number for the web service.

service_name The name assigned to the web service.

service_type The type of the service, for example, RAW, HTTP, XML, SOAP, or DISH.

auth_required (Y/N) Indicates whether all requests must contain a valid user name and password.

secure_required (Y/N) Indicates whether insecure connections, such as HTTP, are to be accepted, or only secure connections, such as HTTPS.

url_path Controls the interpretation of URLs.

user_id If authentication is enabled, identifies the user, or group of users, that have permission to use the service. If authentication is disabled, specifies the account to use when processing requests.

parameter A prefix that identifies the SOAP services to be included in a DISH service.

statement A SQL statement that is always executed in response to a request. If NULL, arbitrary statements contained in each request are executed instead. Ignored for services of type DISH.

CHAPTER 10

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.
- Transact-SQL system and catalog procedures. For a list of these system procedures, see “Adaptive Server Enterprise system and catalog procedures” on page 884.

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 10-1 for an explanation of syntax variations.

Table 10-1: 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.
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 using `SET OPTION DBO.TRUNCATION_LENGTH = 80`. Alternatively, from the dbisql menu select Command > Options and enter a higher value for Limit Display Columns and/or Limit Output Columns.

sa_verify_password 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.

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.

Permissions: None

Side effects: None

Example: The following example creates a function that returns a message if the chosen password can be guessed from knowing the user name:

```
CREATE FUNCTION
DBA.f_verify_pwd(user_name varchar(128), new_pwd
varchar(255))
RETURNS varchar(255)
BEGIN
-- enforcement
IF SIMILAR(new_pwd , user_name) > 50 THEN
RETURN('Password is too much like the user name');
END IF;
-- success
RETURN(null);
END;
ALTER FUNCTION DBA.f_verify_pwd SET HIDDEN;
```

```
GRANT EXECUTE ON DBA.f_verify_pwd TO PUBLIC;
SET OPTION public.verify_password_function =
'DBA.f_verify_pwd';
```

sp_iqaddlogin procedure

Function

Adds a new Sybase IQ user account.

Syntax1

call sp_iqaddlogin (‘*userid*’, ‘*password*’, [*number_of_connections*] [, *password_expiration*])

Syntax2

sp_iqaddlogin ‘*userid*’, ‘*password*’, [*number_of_connections*] [, *password_expiration*]

Syntax3

sp_iqaddlogin *userid*, *password*, [*number_of_connections*] [, *password_expiration*]

Permissions

DBA authority required.

Usage

userid The user’s login name. Login names must conform to the rules for identifiers.

password The user’s password. Passwords must conform to Adaptive Server Anywhere rules for passwords, that is, they must be valid identifiers.

number_of_connections Maximum number of concurrent database connections for the user. Default is 0: Sybase IQ does not enforce a maximum number of connections.

password_expiration Password expiration interval, in days. Must be a value from 0 through 32767. Default is 0: the password does not expire.
You cannot set a password expiration for the user DBA.

A userid/password 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.

“sp_iqmodifyadmin procedure” on page 806

GRANT statement on page 559

Chapter 12, “Managing User IDs and Permissions,” in *Sybase IQ System Administration Guide*

Description

Adds a new Sybase IQ user account, specifies the number of concurrent logins the user might have and the password expiration interval, and adds the user to the IQ_USER_LOGIN_INFO_TABLE system table. If the user already has a user ID for the database but is not in IQ_USER_LOGIN_INFO_TABLE (for example, if the user ID was added using the GRANT CONNECT statement or Sybase Central), sp_iqaddlogin adds the user to the table.

By default, you can add users with sp_iqaddlogin only on a multiplex write server. To enable sp_iqaddlogin on query servers, you must set the database option MPX_LOCAL_SPEC_PRIV to change the default. For details, see "MPX_LOCAL_SPEC_PRIV option" on page 123.

You must add users with sp_iqaddlogin to manage them with the Sybase IQ Login Management facility.

Errors

The following errors might occur. Causes are listed after each error.

```
Permission denied:  you do not have permission to
execute the procedure sp_iqaddlogin.
```

Cause: A user without DBA role tried to execute sp_iqaddlogin.

```
RAISERROR executed: User <loginname> already exists.
```

Cause: The message appears if a user being created already exists for the database.

```
RAISERROR executed: "NUMBER_OF_CONNECTIONS must be
greater than or equal to zero and less than of equal to
32767."
```

Cause: Number of connections value was something other than 0 through 32767.

```
RAISERROR executed: "PASSWORD_EXPIRATION must be
greater than or equal to zero and less than or equal to
32767."
```

Cause: Number of days for password expiration was something other than 0 through 32767.

Examples

The following stored procedure calls add the user rose and allow that user 5 concurrent connections with a password irk324 that expires in 180 days.

```
call sp_iqaddlogin ('rose','irk324',5,180)
```

```
sp_iqaddlogin 'rose','irk324',5,180
```

sp_iqcheckdb procedure

Function

Checks validity of the current database and optionally repairs indexes and allocation problems.

This stored procedure 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, if the -iqdroplks server switch is used. 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 any errors are detected.

The sp_iqcheckdb stored procedure also allows you to check the consistency of, and optionally repair, 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 repair mode. 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 } | repair

target:
[main | local | indextype *index-type* […]] database | database resetclocks |
{ [indextype *index-type*] […] table *table-name* | index *index-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*]'

Repair mode

sp_iqcheckdb 'repair *target* [resources *resource-percent*]'

Usage

main All tables and indexes checked are from the IQ Store. In a multiplex, they are from the shared IQ Store.

local All tables and indexes checked are from the local IQ Store on a particular query server in a multiplex.

index-type The *index-type* parameter is 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 The *index-name* parameter 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 The *table-name* parameter 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 quotes, as shown in this example:

```
sp_iqcheckdb 'check index "dbo.ss tab.i2" resources 75'
```

---

resource-percent The input parameter *resource-percent* must be an integer greater than 0. 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, then 1 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 1. The default value of *resource-percent* is 100.

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 check, verify, and allocation mode by starting the database with the -iqdroplks server switch.

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 four 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, or a specific table; repairs the free list if the -iqdroplks server switch is specified. Does not check index consistency.

sp_iqcheckdb cannot check or repair all allocation problems, if you specify the name of a single index, index type, or table in the input parameter string.

When to run in allocation mode:

- After forced recovery, run sp_iqcheckdb with the -iqdroplks server switch to reset the allocation map (must use database as the target)
- To check for 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. The resetclocks option is used in conjunction with forced recovery to convert a multiplex query server to a write server. 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 command “allocation database”. resetclocks does not require the -iqdroplks server start-up switch. The syntax of the resetclocks command is:

```
sp_iqcheckdb 'allocation database resetclocks'
```

Check mode Checks allocation with index information; performs quick index checks for the entire database, a specific index, a specific index type, or a specific table. Detects all types of allocation problems and most types of index inconsistencies.

Run in check mode if metadata, null count, or distinct count errors are returned when running a query.

Verify mode Checks allocation with index information; performs detailed index checks for the entire database, a specific index, a specific index type, or a specific table. Detects all types of allocation problems and all types of index inconsistencies.

Run in verify mode if metadata, null count, or distinct count errors are returned when running a query.

Repair mode Performs a detailed check and repair of all indexes, a specific index, or a specific table. Does not check allocation.

Run in repair mode if index errors are reported in sp_iqcheckdb check or verify mode.

Note sp_iqcheckdb does not check referential integrity or repair referential integrity violations.

See Chapter 2, "System Recovery and Database Repair" in the *Sybase IQ Troubleshooting and Recovery Guide* for details on using sp_iqcheckdb and more information on checking database consistency.

Examples

The following examples illustrate the use of the sp_iqcheckdb procedure.

In this example, sp_iqcheckdb checks the allocation for the entire database:

```
sp_iqcheckdb 'allocation database'
```

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'
```

You can combine all modes, except for repair mode, and run multiple checks on a database in a single session. In the following example, sp_iqcheckdb performs a quick check of 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 verify index i1
allocation database resources 50'
```

This example checks all indexes of the type FP in the database:

```
sp_iqcheckdb 'check indextype FP database'
```

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'
```

DBCC performance

The execution time of DBCC varies according to 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.

Table 10-2 summarizes the actions and output of the four sp_iqcheckdb modes.

Table 10-2: 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
repair	All index errors	Repair statistics	15+GB per hour*

* The processing time of sp_iqcheckdb repair mode depends on the number of errors repaired.

Output

The output of sp_iqcheckdb contains summary results, errors, informational statistics, and repair statistics, depending on the execution mode. 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 are displayed only if errors are detected.

Repair statistics are displayed only in repair mode and only if repairs are actually made. Asterisks (*****) indicate repairs that were made, not errors. If sp_iqcheckdb encounters errors and makes repairs, some of the statistics reported by DBCC in repair mode might be inaccurate.

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           |*****
   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                |                          |
=====================================|==========================|=====
```

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 start-up options that have been changed from the default.

Syntax

sp_iqcheckoptions

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.

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 start-up options.

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 ASA database options. Sybase IQ modifies some ASA 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 10-3: 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       Min_NLPDJ_Table_Size 1000000         10000           Permanent
PUBLIC    Min_NLPDJ_Table_Size 1000000         10000           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      Min_NLPDJ_Table_Size 1000000         10000           Permanent
PUBLIC    Min_NLPDJ_Table_Size 1000000         10000           Permanent
```

```
joel      Thread_count         25              0               Temporary
```

sp_iqcolumn procedure

Function	Displays columns in a database and information about them.
Syntax1	**sp_iqcolumn** ([*table_name*],[*table_owner*])
Syntax2	**sp_iqcolumn** [table_name=*'tablename'*],[table_owner=*'tableowner'*]
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 *'tablename'* and *'tableowner'* in single quotes.
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 parameters chooses the columns from a unique table, if that table exists. Specifying no parameters returns all columns for all tables in a database. This procedure does not return column information for system tables.

Table 10-4: 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 Temp Store, MAIN = IQ Store, LOCAL = IQ Local Store, SYSTEM = Catalog Store
remarks	User comments added with the COMMENT statement
check	the check constraint expression

Examples

The following variations in syntax both return all of the columns in the table department:

```
sp_iqcolumn department

call sp_iqcolumn (table_name='department')
```

table_ name	table_ owner	column_ name	domain _name	width	scale	nulls	default	cardi- nality	est_ cardi- nality	location	remarks	check
department	DBA	dept_id	unsigned int	4	0	N	N	5	5	(NULL)	(NULL)	
department	DBA	dept_ name	char	40	0	N	N	0	5	(NULL)	(NULL)	
department	DBA	dept_head _id	unsigned int	4	0	Y	N	5	5	(NULL)	(NULL)	

The following variations in syntax both return all of the columns in all of the tables owned by table owner DBA. For brevity, some rows have been omitted from the results shown:

```
sp_iqcolumn table_owner='DBA'

sp_iqcolumn NULL,DBA
```

table_name	table_owner	column_name	domain_name	width	scale	nulls	default	cardinality	est_cardinality	location	remarks	check
contact	DBA	id	unsigned int	4	0	N	(NULL)	60	60	(NULL)	(NULL)	
contact	DBA	last_name	char	15	0	N	(NULL)	0	60	(NULL)	(NULL)	
...	...	...	...	...	...	...	(NULL)	...	...	...	...	...
contact	DBA	phone	char	10	0	Y	(NULL)	0	59	(NULL)	(NULL)	
contact	DBA	fax	char	10	0	Y	(NULL)	0	58	(NULL)	(NULL)	
customer	DBA	id	unsigned int	4	0	N	(NULL)	126	126	(NULL)	(NULL)	
customer	DBA	fname	char	15	0	N	(NULL)	0	116	(NULL)	(NULL)	
...	...	...	...	...	...	...	...	..	...	...	...	...
customer	DBA	phone	char	12	0	N	(NULL)	0	117	(NULL)	(NULL)	
customer	DBA	company_name	char	35	0	Y	(NULL)	0	126	(NULL)	(NULL)	
department	DBA	dept_id	unsigned int	4	0	N	(NULL)	5	5	(NULL)	(NULL)	
department	DBA	dept_name	char	40	0	N	(NULL)	0	5	(NULL)	(NULL)	
department	DBA	dept_head_id	unsigned int	4	0	Y	(NULL)	5	5	(NULL)	(NULL)	
...	...	...	...	...	...	...	...	...	...	...	...	...

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.

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. In Java applications, specify Sybase IQ-specific connection properties from TDS clients in the Remote PWD field. For details, see "Using the RemotePWD parameter" in the *Sybase IQ System Administration Guide*.

Table 10-5: 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.
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 IDof 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.

Column name	Description
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 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.
satoiq_count	An internal counter used to display the number of crossings from the ASA 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 ASA side of the Sybase IQ engine. This might 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.
NodeAddr	The node for the client in a client/server connection.
LastIdle	The number of ticks between requests.
Dbremote	A bit data column that indicates the transaction is an internal transaction used to replicate multiplex version information between a query server and the write server within a multiplex database.

Example

The following is an example of sp_iqconnection output:

```
ConnHandle     Name Userid       LastReqTime                 ReqType                 IQCmdType
========== ======== ====== ======================== =================== ====================
 419740283    red2  DBA  2006-06-262006-06-26 15:54:54.605      STMT_EXECUTE_IMM            INSERT
 640038605    blue1  DBA  2006-06-26 13:32:42.505         CURSOR_PREFETCH                  NONE
2094200996           DBA  2006-06-26 13:30:27.486    STMT_EXECUTE_ANY_IMM                  NONE
 954498130  fromSCJ  DBA  2006-06-26 15:55:02.787752            STMT_DROP                  NONE
 167015670    blue2  DBA  2006-06-26 13:45:50.232752            STMT_DROP                  NONE
1306718536           DBA  2006-06-26 15:08:36.716    STMT_EXECUTE_ANY_IMM                  NONE
1779741471  ntJava2  DBA  2006-06-26 15:54:58.558752            STMT_DROP                  NONE
 710225777      nt1  DBA  2006-06-26 15:56:02.729             CURSOR_OPEN  IQUTILITYOPENCURSOR


LastIQCmdTime           IQCursors LowestIQCursorState IQthreads TxnID      ConnCreateTime
======================= ========= ================== ========= ===== =======================
2006-06-26 15:54:54.630     1            EXECUTED         7     10701  2006-06-26 13:17:27.599
2006-06-26 13:32:42.295     1            FETCHING         2     10568  2006-06-26 13:21:19.953
2006-06-26 13:30:27.548     0                NONE         1     10604  2006-06-26 13:24:35.145
2006-06-26 15:55:02.590     0                NONE         1     10619  2006-06-26 13:31:26.001
2006-06-26 13:45:50.225     0                NONE         1     10678  2006-06-26 13:35:01.160
2006-06-26 15:09:30.320     0                NONE         1     16687  2006-06-26 13:37:50.814
2006-06-26 15:54:58.553     0                NONE         1     10676  2006-06-26 13:43:57.907
2006-06-26 15:56:02.755     0                NONE         1     10699  2006-06-26 14:05:15.748
```

```
TempTableSpaceKB TempWorkSpaceKB IQconnID satoiq_count iqtosa_count CommLink   NodeAddr  LastIdle
================ =============== ======== ============ ============ ======== ==========  =======
           68736             680       14           82         2031 TCPIP    157.133.82.17  9905
               0          102592       17           76          360 local                    606
               0               0       18          397          688 TCPIP   157.133.83.151  8322
               0               0       20          709         1541 TCPIP   157.133.83.151  5378
               0             128       21          131         2082 local                   5122
               0               0       23        18313          821 TCPIP   157.133.83.151 10000
               0               0       24          994         1667 TCPIP   157.133.83.151  1467
               0               0       28          900          478 TCPIP   157.133.83.151  5473
```

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*)

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
- Owner
- Candidate key index
- Primary key or Unique
- Primary key columns

- Foreign table
- Owner
- Foreign key role name
- Enforced status (“Y” for enforced, “N” for unenforced)
- Foreign key index
- Foreign key columns
- Location (“TEMP,” “MAIN,” “LOCAL,” or “SYSTEM”)

sp_iqcontext procedure

Function

Tracks and displays, by connection, information about statements 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 for that connection only.

Description

sp_iqcontext lets the DBA determine what statements are running on the system at any given moment, and to identify the user and connection that issued the statement. With this information, you can use this utility to quickly:

- 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 it is possible for Sybase IQ to 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 10-6: sp_iqcontext columns

Column name	Description
ConnOrCursor	CONNECTION or CURSOR.
ConnHandle	The ID number of the connection.
Name	The name of the server.
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 a user's queries are queued for execution. In the range of allowed values, 1 indicates high priority, 2 (the default) medium priority, and 3 low priority. This value is set per user with the database option IQGOVERN_PRIORITY. For details, see *Sybase IQ Reference Manual*.
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 2006-06-04 09:24:17.000 4 NO COMMAND
CURSOR 701773517 dba7 DBA 1 0 1324 2006-06-04 09:24:46.000 4 2 select * from foo1
CURSOR 701773517 dba7 DBA 2 0 1324 2006-06-04 09:24:47.000 4 2 select a from foo1
...
CURSOR 701773517 dba7 DBA 6 0 1324 2006-06-04 09:24:47.000 4 2 select e from foo1
CONNECTION 1271624950 dba7 DBA 0 12 1377 2006-06-04 09:24:12.000 3 2 sp_iqcheckdb
CONNECTION 1841476383 dba7 DBA 10 1 1337 2006-06-04 09:24:19.000 5 2 call sp_iqcontext()
CURSOR 1841476383 dba7 DBA 1 0 1337 2006-06-04 09:24:47.000 5 2 select * from foo
...
CURSOR 1841476383 dba7 DBA 10 0 1337 2006-06-04 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.) Note that 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_iqcursorinfo procedure

Function

Displays detailed information about cursors currently open on the server.

Syntax

sp_iqcursorinfo [*cursor-name*] [, *conn-handle*]

Permissions

DBA permission required.

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 10-7: 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

See also

In Chapter 6, "SQL Statements": DECLARE CURSOR statement [ESQL] [SP] on page 516 and DECLARE CURSOR statement [T-SQL] on page 522

In Chapter 6, "SQL Statements": UPDATE (positioned) statement [ESQL] [SP] on page 664 and DELETE (positioned) statement [ESQL] [SP] on page 527

In Chapter 2, "Database Options": "FORCE_NO_SCROLL_CURSORS option" on page 79 and "FORCE_UPDATABLE_CURSORS option" on page 80

"Using cursors in procedures" in Chapter 8, "Using Procedures and Batches" in the *Sybase IQ System Administration Guide*

"Cursors in transactions" in Chapter 10, "Transactions and Versioning" in the *Sybase IQ System Administration Guide*

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 10-8: 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.
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
-----------------------------------------------------------------
2006-06-26 15:24:36.000          19      100000000     200000000
2006-06-26 15:38:38.000       20000      200000000

NumDelete     NumInsert      RWTabOwner      RWTabName         CmdLine
-------------------------------------------------------------------------
 20000000    3000000000             DBA          test1    call proc1()
                                                          call proc2()
```

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 None required.

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 10-9: 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 rowid', dbo, ALL	No rows returned, as the data type rowid is owned by SYS

See also

"SYSUSERTYPE system table" on page 734

"SYSDOMAIN system table" on page 697

CREATE DOMAIN statement on page 456

Chapter 4, "SQL Data Types"

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 10-10: 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)
```

sp_iqdbsize procedure

Function

Displays the size of the current database.

Syntax

sp_iqdbsize(

[**main** | **local**]

)

See also

“Specifying page size” in “Overview of memory use” in Chapter 5, “Managing System Resources” in the *Sybase IQ Performance and Tuning Guide*

“Working with database objects” in Chapter 5, “Working with Database Objects” in the *Sybase IQ System Administration Guide*

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). If a multiplex database, the default is main, the size of the shared IQ Store. The optional parameter local specifies only information about the IQ Local Store owned by the query server.

Table 10-11: 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 5, “Managing System Resources” in the *Sybase IQ 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

This example displays size information for the database asiqdemo.

```
sp_iqdbsize
```

```
Database
PhysicalBlocks KBytes Pages CompressedPages NBlocks CatalogBlocks
============== ====== ===== =============== ======= =============
/system1/sybase/ASIQ-12_7/demo/asiqdemo.db
          1280    522   688             257    1119             18
```

sp_iqdbspace procedure

Function

Displays detailed information about each dbspace.

Syntax

sp_iqdbspace [*dbspace-name*]

Permissions

DBA authority required.

See also

- "sp_iqdbspaceinfo procedure" on page 769, "sp_iqindexinfo procedure" on page 790, and "sp_iqrelocate procedure" on page 822
- Chapter 5, "Working with Database Objects" in the *Sybase IQ System Administration Guide*

Description

The sp_iqdbspace stored procedure displays the usage, properties, and types of data on each dbspace. You can use this information to determine whether data must be relocated, and for data that has been relocated, whether the old versions have been deallocated.

sp_iqdbspace output fields include the dbspace name, path, type, mode, percent used, size, reserve, writes per stripe, block type, first block, and last block.

Name Name of the dbspace in the SYSFILE system table and as specified in the CREATE DBSPACE statement. Dbspace names are case sensitive for databases created with CASE RESPECT and case insensitive for databases created with CASE IGNORE.

Path Location of the dbspace file or raw partition.

Segment Type Type of dbspace: MAIN, TEMPORARY, or LOCAL.

RWMode Mode of the dbspace: readwrite (RW), relocate (RR), or readonly (RO).

Usage Percent of dbspace currently in use.

DBSSize Current size of the dbspace 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 the dbspace.

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 10-12 for identifier values.

FirstBlk First IQ block number assigned to the dbspace.

LastBlk Last IQ block number assigned to the dbspace.

Table 10-12 lists the values of the block type identifiers.

Table 10-12: sp_iqdbspace block types

Identifier	Block Type
A	Active Version
B	Backup Structures
C	Checkpoint Log
D	Database Identity
F	Freelist
H	Header Blocks of the free list
I	Index advice storage
M	Multiplex CM
O	Old Version
R	Readonly Freelist
X	Drop at checkpoint

Examples

The following output displays information about dbspaces.

```
sp_iqdbspace;
```

Name	Path	Segment Type	RW Mode	Usage	DBS Size	Reserve	Stripe Size	Blk Types	First Blk	Last Blk
IQ__SYSTEM_MAIN	D:\IQ\dbspacedb.iq	MAIN	RW	24	10M	100M	8K	1H,64F,32D,62A,20X,128M	1	1280
dbspacedb2	D:\IQ\dbspacedb.iq2	MAIN	RW	9	10M	20M	8K	1H,32F,56A,19X	1045440	1046719
dbspacedb3	D:\IQ\dbspacedb.iq3	MAIN	RW	12	10M	40M	8K	1H,32F,59A,49X	2090880	2092159
IQ_SYSTEM_TEMP	dbspacedb.iqtmp	TEMPORARY	RW	8	10M	10M	8K	1H,64F,12A,20X	1	1280

The following output displays information about dbspaces with three different readwrite modes (the RWMode column):

```
sp_iqdbspace;
```

Name	Path	Segment Type	RW Mode	Usage	DBS Size	Reserve	Stripe Size	Blk Types	First Blk	Last Blk
IQ__SYSTEM_MAIN	D:\IQ\dbspacedb.iq	MAIN	RR	4	15M	95M	8K	1H,64F,62A	1	1920
dbspacedb2	D:\IQ\dbspacedb.iq2	MAIN	RO	5	10M	20M	8K	1H,32F,56A	1045440	1046719
dbspacedb3	D:\IQ\dbspacedb.iq3	MAIN	RW	25	10M	40M	8K	1H,64F 33R,32D,59A,128M	2090880	2092159
IQ_SYSTEM_TEMP	dbspacedb.iqtmp	TEMPORARY	RW	8	10M	10M	8K	1H,64F,32A	1	1280

sp_iqdbspaceinfo procedure

Function

Displays the number of blocks used per index per main or local dbspace for one or all dbspaces.

Syntax

sp_iqdbspaceinfo ['*dbspace-name-pattern*'] [,'local']

Permissions

DBA authority required.

See also

- "sp_iqdbspace procedure" on page 766, "sp_iqindexinfo procedure" on page 790, "sp_iqspaceinfo procedure" on page 829, and "sp_iqrelocate procedure" on page 822
- Chapter 5, "Working with Database Objects" in the *Sybase IQ System Administration Guide*

Usage

dbspace-name-pattern If specified, sp_iqdbspaceinfo displays output only for dbspaces that match LIKE pattern. sp_iqdbspaceinfo displays all dbspace names, if *dbspace-name-pattern* is not specified.

local The local keyword is specified to enable the display of objects in the multiplex local IQ store. By default on a query server, sp_iqdbspaceinfo displays information about the shared main IQ store on a query server.

Description

The sp_iqdbspaceinfo stored procedure shows the DBA which objects reside on each dbspace. The DBA can use this information to determine which objects must be relocated before a dbspace can be dropped.

The results of sp_iqdbspaceinfo are displayed from the point of view of the version seen by the transaction running the command. Blocks used by other versions are not shown.

Table 10-13: sp_iqdbspaceinfo columns

Column name	Description
dbspace_name	Name of the dbspace
Object	Table, index, or join index name
MinBlk	First block used by this object on this dbspace
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
ObjSize	Size of data for this object on this dbspace
DBSpSz	Size of the dbspace

Example

The following output displays information about all main dbspaces.

```
sp_iqdbspaceinfo;
```

dbspace_name	Object	MinBlk	MaxBlk	ObjSize	DBSpSz
dbspacedb2	t2	1045495	1045495	8K	10M
dbspacedb2	t2.DBA.t2c1hng	1045537	1045553	136K	10M
dbspacedb3	t1	2090913	2091321	200K	10M
dbspacedb3	t1.DBA.ASIQ_IDX_T429_C1_FP	2090914	2091316	288K	10M
dbspacedb3	t1.DBA.t1c1hg	2090931	2091280	304K	10M
dbspacedb3	t2	2090930	2091261	192K	10M
dbspacedb3	t2.DBA.ASIQ_IDX_T430_C1_FP	2091027	2091277	288K	10M

The following output displays information about a specific dbspace in the database:

```
sp_iqdbspaceinfo IQ_SYSTEM_MAIN;
```

dbspace_name	Object	MinBlk	MaxBlk	ObjSize	DBSpSz
IQ_SYSTEM_MAIN	t1	82	125	40K	10M
IQ_SYSTEM_MAIN	t1.DBA.ASIQ_IDX_T429_C1_FP	109	322	136K	10M
IQ_SYSTEM_MAIN	t1.DBA.t1c1hg	127	305	152K	10M
IQ_SYSTEM_MAIN	t2	84	107	32K	10M
IQ_SYSTEM_MAIN	t2.DBA.ASIQ_IDX_T430_C1_FP	126	321	136K	10M

sp_iqdbstatistics procedure

Function — Reports results of the most recent sp_iqcheckdb.

Syntax — **sp_iqdbstatistics**

See also — For more information on the use of sp_iqcheckdb and the interpretation of the sp_iqcheckdb output, see Chapter 2, “System Recovery and Database Repair” in the *Sybase IQ Troubleshooting and Recovery Guide*.

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               |                          |
====================================|==========================|=====
```

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.
Usage	**userid** User ID of the user to drop.
See also	“sp_iqaddlogin procedure” on page 741 REVOKE statement on page 628 Chapter 12, “Managing User IDs and Permissions” in *Sybase IQ System Administration Guide*
Description	sp_iqdroplogin drops the specified user, and removes the user from the IQ_USER_LOGIN_INFO_TABLE. By default, you can only drop users with sp_iqdroplogin on a multiplex write server. To enable sp_iqdroplogin on query servers, you must set the database option MPX_LOCAL_SPEC_PRIV to change the default. For details, see “MPX_LOCAL_SPEC_PRIV option” on page 123.

Errors	The following errors may occur. Causes are listed after each error.
	`Permission denied: You do not have permission to execute the procedure sp_iqdroplogin.`
	Cause: A user without DBA role tried to execute sp_iqdroplogin.
	`RAISERROR executed: User <loginname> does not exist.`
	Cause: The message appears if the user tries to drop a nonexistent login.

Examples	The following stored procedure calls remove the user rose.
	`sp_iqdroplogin 'rose'`
	`sp_iqdroplogin rose`
	`call sp_iqdroplogin ('rose')`

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*)

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). Table 10-14 lists the sp_iqestjoin parameters.

Table 10-14: 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 ( 'customer', 1500000, 'orders',
15000000, 'one>>many', 65536 )
```

Cases	Indexsize	Create time	Msg
Table1:customer			
Rows: 1500000			
Columns:			
8			
Width:			
223			
Table2: orders			
Rows: 15000000			
Columns:			
9			
Width:			
134			

Cases	Indexsize	Create time	Msg
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*)

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 10-15 lists the sp_iqestdbspaces parameters.

Table 10-15: 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
```

dbspaces	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*)

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 10-16 lists the sp_iqestspace parameters.

Table 10-16: 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

None required.

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 10-17: 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

See also

"SYSEVENT system table" on page 697

"SYSEVENTTYPE system table" on page 698

CREATE EVENT statement on page 458

Chapter 18, "Automating Tasks Using Schedules and Events" in the *Sybase IQ System Administration Guide*

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 10-18: 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)
```

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

None required.

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 10-19: 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 `"Object not found"` is returned.

Table 10-20: 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*

See also

In Chapter 9, "System Tables": "SYSPROCEDURE system table" on page 718, "SYSTABLE system table" on page 728, "SYSEVENT system table" on page 697, "SYSUSERTYPE system table" on page 734, "SYSCOLUMN system table" on page 694, "SYSCONSTRAINT system table" on page 696, "SYSINDEX system table" on page 703, "SYSPROCPARM system table" on page 720, "SYSDOMAIN system table" on page 697

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 10-21: 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 10-22: 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

Note For descriptions of the individual output columns listed in Table 10-22, refer to the descriptions of the following stored procedures:

- table: “sp_iqtable procedure” on page 839
- column: “sp_iqcolumn procedure” on page 751
- index: “sp_iqindex and sp_iqindex_alt procedures” on page 786
- constraint: “sp_iqconstraint procedure” on page 756
- join index: “sp_iqjoinindex procedure” on page 795
- view: “sp_iqview procedure” on page 848 and the Adaptive Server Enterprise catalog procedure sp_columns (for view columns)
- stored procedure and function: “sp_iqprocedure procedure” on page 813 and “sp_iqprocparm procedure” on page 816 (for procedure parameters)
- event: “sp_iqevent procedure” on page 776
- data type: “sp_iqdatatype procedure” on page 762

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 ASE in terms of types of objects supported and the namespace of objects. In ASE, 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 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 ASE, the types of objects supported by and the syntax of Sybase IQ sp_iqhelp are different from the supported objects and syntax of ASE 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 Remarks  table_constraints
========== =========== ==========  ======= =======  =============

 sale      DBA          IQ           Main    (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    remarks    check
  ==============     =======    =====
  0                  (NULL)     (NULL)
  0                  (NULL)     (NULL)
  0                  (NULL)     (NULL)
  0                  (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 customer
                            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  remarks
  ======= ========
  (NULL)   (NULL)
```

sp_iqindex and sp_iqindex_alt procedures

Function — Lists indexes and information about them.

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'*]

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 department,NULL,DBA`.

Syntax2 The parameters can be specified 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 10-23: 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 Temp Store, MAIN = IQ Store, LOCAL = IQ Local Store, SYSTEM = Catalog Store
remarks	User comments added with the COMMENT statement

The sp_iqindex format always produces one line per index. The sp_iqindex_alt format 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 dept_id:

```
call sp_iqindex (NULL,'dept_id')
sp_iqindex column_name='dept_id'
```

table_ name	table_ owner	column_ name	index_ type	index_name	unique_ index	location	remarks
department	DBA	dept_id	FP	ASIQ_IDX_T201_C1_FP	N	Main	(NULL)
department	DBA	dept_id	HG	ASIQ_IDX_T201_C1_HG	U	Main	(NULL)
employee	DBA	dept_id	FP	ASIQ_IDX_T202_C5_FP	N	Main	(NULL)

The following variations in syntax both return all indexes in the table department that is owned by table owner DBA:

```
sp_iqindex department,NULL,DBA
sp_iqindex table_name='department',table_owner='DBA'
```

table_ name	table_ owner	column_ name	index_ type	index_name	unique_ index	location	remarks
department	DBA	dept_head_id	FP	ASIQ_IDX_T201_C3_FP	N	Main	(NULL)
department	DBA	dept_id	FP	ASIQ_IDX_T201_C1_FP	N	Main	(NULL)
department	DBA	dept_id	HG	ASIQ_IDX_T201_C1_HG	U	Main	(NULL)
department	DBA	dept_name	FP	ASIQ_IDX_T201_C2_FP	N	Main	(NULL)

The following variations in syntax for sp_iqindex_alt both return indexes on the table employee 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 employee,city
sp_iqindex_alt table_name='employee',
               column_name='city'
```

table_ name	table_ owner	column_ name	index_ type	index_name	unique_ index	remarks
employee	DBA	city	FP	ASIQ_IDX_T452_C7_FP	N	(NULL)
employee	DBA	city	HG	emp_loc	N	(NULL)
employee	DBA	state	HG	emp_loc	N	(NULL)

Notice that the output from the sp_iqindex procedure for the same table and column is slightly different:

```
sp_iqindex employee,city
sp_iqindex table_name='employee',column_name='city'
```

table_ name	table_ owner	column_ name	index_ type	index_name	unique_ index	location	remarks
employee	DBA	city	FP	ASIQ_IDX_T452_C7_FP	N	Main	(NULL)
employee	DBA	city,state	HG	emp_loc	N	Main	(NULL)

sp_iqindexadvice procedure

Function — Displays stored index advice messages. Optionally clears advice storage.

Syntax — **sp_iqindexadvice** ([*resetflag*])

Permissions — This procedure is owned by dbo. Users without DBA authority must be granted execute permission for the stored procedure in order to run it.

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.

See also — “INDEX_ADVISOR option” on page 85

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.

INDEX_ADVISOR columns are described as follows:

- Advice – Unique advice message
- NInst – Number of instances of message
- LastDT – Last Date/Time advice was generated

Examples

Table 10-24 illustrates sample output from the sp_iqindexadvice procedure.

Table 10-24: Sample sp_iqindexadvice output

Advice	NInst	LastDT
Add a CMP index on DBA.tb (c2, c3) Predicate: (tb.c2 = tb.c3)	2073	2006-04-07 16:37:31.000
Convert HG index on DBA.tb.c4 to a unique HG	812	2006-04-06 10:01:15.000
Join Key Columns DBA.ta.c1 and DBA.tb.c1 have mismatched data types	911	2006-02-25 20:59:01.000

sp_iqindexfragmentation procedure

Function

Reports information about the amount of empty space within the btrees, garrays,and bitmaps in Sybase IQ indexes.

Syntax

dbo.sp_iqindexfragmentation ('*target* ')

target: **table** *table-name* (**index** *index-name* (...))

Permissions

This procedure is owned by dbo. Users without DBA authority need to be granted execute permission in order to run it.

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 within the specified table. You may specify multiple indexes within the table, but must repeat the index keyword with each index specified.

Example

The following procedure 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	Fill factor percent
dba.customers.cidhg	HG	3	75
SQLCODE	0		
Fill Percent	btree pages	garray pages	bitmap pages
0 - 10%	0	0	0

Index	Index type	Btree node pages	Fill factor percent
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 5	138	3	64
81 - 90%	24	122	14
91 - 100%	18	1	0

According to this output, of the 182 btree 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 btree 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.

sp_iqindexinfo procedure

Function

Displays the number of blocks used per index per main or local dbspace for a given object.

Syntax

sp_iqindexinfo '{ **database | local**
| [**table** *table-name* | **index** *index-name*] [...] }
[**resources** *resource-percent*]'

Permissions

DBA authority required.

See also

- "sp_iqdbspace procedure" on page 766, "sp_iqdbspaceinfo procedure" on page 769, "sp_iqspaceinfo procedure" on page 829, and "sp_iqrelocate procedure" on page 822
- Chapter 5, "Working with Database Objects" in the *Sybase IQ System Administration Guide*

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 or local 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.

The LOCAL keyword is specified as a target to enable the display of objects in the IQ Local Store. By default in a multiplex database, sp_iqindexinfo displays information about the shared IQ Store on a query server. If individual tables or indexes are specified, then the store to display is selected automatically. You cannot specify targets from both the shared IQ Store and IQ Local Stores.

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

The sp_iqindexinfo stored procedure 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 displayed from the point of view of the version seen by the transaction running the command. Blocks used by other versions are not shown.

Table 10-25: 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
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

The following command 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

Because you cannot specify targets from both the shared IQ Store and IQ Local Stores, the following command returns an error, if local_tab1 is a local table and main_tab1 is a shared IQ table:

```
sp_iqindexinfo 'table local_tab1 table main_tab1'
```

sp_iqindexmetadata procedure

Function

Displays the index metadata for the 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 or EXECUTE permission required.

See also

- "sp_iqindex and sp_iqindex_alt procedures" on page 786, "sp_iqindexfragmentation procedure" on page 789, "sp_iqindexinfo procedure" on page 790, and "sp_iqindexsize procedure" on page 793
- Chapter 5, "Working with Database Objects" in the *Sybase IQ System Administration Guide*

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 10-26: 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',	''

sp_iqindexsize procedure

Function

Gives the size of the specified index.

Syntax

sp_iqindexsize [[*owner*.] *table*.] *index_name*

Description

Table 10-27: 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 Btree (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 Btree (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	employee.ASIQ_IDX_T452_C19_FP	FP	Total	288	4	2
DBA	employee.ASIQ_IDX_T452_C19_FP	FP	vdo	0	0	0
DBA	employee.ASIQ_IDX_T452_C19_FP	FP	bt	0	0	0
DBA	employee.ASIQ_IDX_T452_C19_FP	FP	garray	0	0	0
DBA	employee.ASIQ_IDX_T452_C19_FP	FP	bm	136	2	1
DBA	employee.ASIQ_IDX_T452_C19_FP	FP	barray	152	2	1
DBA	employee.ASIQ_IDX_T452_C19_FP	FP	dpstore	0	0	0

Username	Indexname	Type	Info	KBytes	Pages	Compressed Pages
DBA	employee.ASIQ_IDX_T452_C19_FP	FP	largelob	0	0	0

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

None required.

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 10-28: 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

In Chapter 9, “System Tables”: “SYSIQJOININDEX system table” on page 710, “SYSIQJOINIXTABLE system table” on page 712, “SYSIQJOINIXCOLUMN system table” on page 711

CREATE JOIN INDEX statement on page 481

Chapter 6, “Using Sybase IQ Indexes” in the *Sybase IQ System Administration Guide*

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 10-29: 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

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 remarks
t1_t2_t3_join  DBA     t1              DBA              n1
=          t2               DBA               n1                NATURAL
Y    (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 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    (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 remarks
t1_t2_t3_join  DBA     t2              DBA              name
=         t3               DBA               name              NATURAL
Y    (NULL)
```

sp_iqjoinindexsize procedure

Function — Gives the size of the specified join index.

Syntax — **sp_iqjoinindexsize** (*join_index_name*)

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 10-30: 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_iqlistexpiredpasswords procedure

Function

Lists users with expired passwords.

Syntax

sp_iqlistexpiredpasswords ['userid']

Usage

userid Returns a row if the specified user's password has expired, or no results if the password has not expired.

By default, this procedure returns a list of users whose passwords have expired.

Permissions

DBA authority required.

See also

"sp_iqmodifyadmin procedure" on page 806

"sp_iqmodifylogin procedure" on page 809

Errors

The following error may occur. Cause is listed after the error.

```
Permission denied: You do not have permission to execute
the procedure "sp_iqlistexpiredpasswords".
```

Cause: A user without DBA role tried to execute sp_iqlistexpiredpasswords.

Example

```
call sp_iqlistexpiredpasswords
```

Expired_Users
jack
jill

sp_iqlistlockedusers procedure

Function

Lists user IDs that are locked out of the database.

Syntax

sp_iqlistlockedusers ['*userid*'] ['*server-name* | all servers']

Permissions

DBA authority required.

Usage

userid If specified, lists the lock status for only the specified user.

If not specified, or null, lists the lock status for all users.

server-name | all servers If specified, lists user IDs locked out of the named server, or, if all servers is specified, user IDs locked on a server-by-server basis. The server name argument is valid only in multiplex environments, and the specified server name must be a valid server name in the IQ_MPX_INFO system table.

If not specified, lists user IDs that are locked by default on a global basis. In a multiplex environment, the global default lock status may be overidden on a server-by-server basis, so user IDs listed without using the server name argument may not be locked out of all servers. In a multiplex environment, Sybase recommends that you specify all servers in order to list which users are effectively locked out of each server.

To display per-server user settings in Sybase Central, right-click the name of a multiplex server and choose Properties from the drop-down. Then choose the Login Management tab. (The tab displays only if Login Management is enabled for the server.)

See also

“sp_iqlocklogin procedure” on page 802

Errors

The following error may occur. Cause is listed after the error.

```
Permission denied: You do not have permission to execute
the procedure "sp_iqlistlockedusers".
```

Cause: A user without DBA role tried to execute sp_iqlistlockedusers.

Examples

The following lists all users locked by default. (This lock may be overridden on a server-by-server basis.)

```
call sp_iqlistlockedusers
```

Locked_Users	Server
Rose	query1
Rose	query2

The following lists all users effectively locked out of all servers, on a server-by-server basis:

```
sp_iqlistlockedusers null, 'all servers'
```

The following lists all users effectively locked out of server Littleton:

```
sp_iqlistlockedusers null, 'Littleton'
```

The following lists all servers, by server, from which user joe is locked:

```
sp_iqlistlockedusers 'joe', 'all servers'
```

sp_iqlistpasswordexpirations procedure

Function

Lists users, their password creation dates, and how many days the password is valid from the creation date.

Syntax

sp_iqlistpasswordexpirations ['userid']

Usage

userid Lists password creation time and days until password expiration for the specified user.

By default, this procedure returns a list of password creation time and days until password expiration for each user.

Permissions

DBA authority required.

See also

"sp_iqaddlogin procedure" on page 741

"sp_iqmodifylogin procedure" on page 809

Errors

The following error may occur. Cause is listed after the error.

```
Permission denied: You do not have permission to execute
the procedure "sp_iqlistpasswordexpirations".
```

Cause: A user without DBA role tried to execute sp_iqlistpasswordexpirations.

Example

```
sp_iqlistpasswordexpirations
```

```
UserName     Password_Created            Days_til_Expiration
-------      ----------------            -------------------
DBA          2006-01-02 10:13:53.625     Expired
rose         2006-01-05 14:36:38.099     180
jack         2006-01-07 14:44:34.645     0

Password_Expiration_Interval
----------------------------
                          90
                         365
                           0
```

A value of 0 for Days_till_Expiration indicates that the password does not expire.

sp_iqlocklogin procedure

Function: Locks an IQ user account so that the user cannot log in.

Syntax1: **call sp_iqlocklogin** (‘*userid*'[, '*server-name*' | 'all servers'] ‘[lock | unlock]’)

Syntax2: **sp_iqlocklogin** '*userid*'[, '*server-name*' | 'all servers'] ‘[lock | unlock]’

Syntax3: **sp_iqlocklogin** *userid* [, '*server-name*' | 'all servers'] ‘[lock | unlock]’

Permissions: DBA authority required.

Usage: **userid** Name of the account to be locked or unlocked.

server-name If specified, restricts the setting to named server. The server name argument is only valid in multiplex environments, and the specified server name must be a valid server name in the IQ_MPX_INFO system table. If all servers is specified, removes all server level settings and specifies global default setting for all servers in the multiplex.

If not specified, locks the user by default on a global basis. In a multiplex environment, the global default lock status may be overridden on a server by server basis, so user IDs locked by default globally may not be locked out of specific servers.

all servers Removes all server level settings and specifies global setting for all servers in multiplex.

See also: “sp_iqmodifyadmin procedure” on page 806

Description: When Sybase IQ Login Management is enabled, the DBA can use sp_iqlocklogin to prevent or enable a specified user’s ability to log in to the database.

You cannot lock yourself or the DBA account out of the database. Connected users can be locked, but they remain connected. A locked account can be specified as a database owner, and can own objects in any database.

Errors: The following errors may occur. Causes are listed after each error.

```
Permission denied: You do not have permission to execute
the procedure sp_iqlocklogin.
```

Cause: A user without DBA role tried to execute sp_iqlocklogin.

```
RAISERROR executed: You cannot lock yourself out.
```

Cause: User tries to lock him or herself out.

```
RAISERROR executed: "The user DBA cannot be locked."
```

Cause: User tried to lock the DBA user.

```
RAISERROR executed: "Invalid option <what the user
entered> was specified.” "
```

Cause: User typed in invalid input.

```
RAISERROR executed: Server name <server name> not found.
```

Cause: server name value specified did not match a valid server name in IQ_MPX_INFO.

```
RAISERROR executed: Server name parameter not allowed
with this option in non-multiplex mode
```

Cause: procedure was called with server name argument in a non-multiplex environment.

Examples

The following examples lock out the user rose.

```
sp_iqlocklogin 'rose', 'lock'
call sp_iqlocklogin ('rose', 'lock')
```

The following example unlocks the account of the user rose.

```
sp_iqlocklogin rose, 'unlock'
```

The following locks the login for user *fred* on all servers in the multiplex and removes server level settings for user *fred*:

```
sp_iqlocklogin('fred', 'lock', 'all servers')
```

The following locks the login for user *fred* by default globally. Note that in a multiplex, server-level settings can override this global default.

```
sp_iqlocklogin('fred')
```

The following locks the login for user *mary* on server query2:

```
sp_iqlocklogin('mary', 'lock', 'query2')
```

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*])

Usage

Table 10-31 lists the optional sp_iqlocks parameters you can specify to restrict results.

Table 10-31: 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 10-32: sp_iqlocks columns

Column	Description
connection	Connection ID that has the lock.
user_id	User associated with this connection ID.
table_name	Table on which the lock is held.
lock _type	String of characters indicating the type of lock: S – shared. E – exclusive. P – phantom. A – antiphantom. W – write . All locks listed have exactly one of S, E, or W, 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.
lock name	Value identifying the lock.

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 Adaptive Server Anywhere tables, but not on Sybase IQ tables. Unless you have explicitly taken out locks on a table in the Catalog Store, you will never see these types of locks (or their qualifiers T, *, and *nnn*) in a Sybase IQ database. For information on how locking works in Adaptive Server Anywhere tables, see the *Adaptive Server Anywhere SQL User's Guide*.

Examples

The first 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()
connection      user_id table_name      lock_type       lock_name
70187172        'mary'  'DBA.t2'        'S'             (NULL)
604945019       'russ'  'russ.t3'       'S'             (NULL)
604945019       'russ'  'russ.t3'       'W'             (NULL)
1550990889      'DBA'   'dbo.spt_mda'   'S'             (NULL)
1744647885      'DBA'   'russ.t1'       'S'             (NULL)
1744647885      'DBA'   'russ.t1'       'W'             (NULL)
1744647885      'DBA'   'russ.t3'       'S'             (NULL)
2120842322      'shemp' 'shemp.t1'      'S'             (NULL)
2120842322      'shemp' 'shemp.t1'      'W'             (NULL)
```

The next example shows sp_iqlocks with sorting by table name.

```
call sp_iqlocks(0,null,0,'t')
connection      user_id table_name      lock_type       lock_name
70187172        'mary'  'DBA.t2'        'S'             (NULL)
1550990889      'DBA'   'dbo.spt_mda'   'S'             (NULL)
1744647885      'DBA'   'russ.t1'       'S'             (NULL)
1744647885      'DBA'   'russ.t1'       'W'             (NULL)
604945019       'russ'  'russ.t3'       'S'             (NULL)
604945019       'russ'  'russ.t3'       'W'             (NULL)
1744647885      'DBA'   'russ.t3'       'S'             (NULL)
2120842322      'shemp' 'shemp.t1'      'S'             (NULL)
2120842322      'shemp' 'shemp.t1'      'W'             (NULL)
```

sp_iqmodifyadmin procedure

Function	Enables Sybase IQ Login Management for servers and modifies Sybase IQ user account information in the IQ_SYSTEM_LOGIN_INFO_TABLE system table.
Syntax1	**call sp_iqmodifyadmin** (‘{enable | disable}’,)
Syntax2	**call sp_iqmodifyadmin** ('*option* ', *value* [, '*server-name*'])
Syntax3	**sp_iqmodifyadmin** ‘{enable | disable}'
Syntax4	**sp_iqmodifyadmin** ‘*option* ' , [*value*] [, '*server-name*']
Permissions	DBA authority required.
Usage	**enable | disable** Enables or disables Sybase IQ Login Management. Cannot be specified with the *server-name* argument.

option Name of the option to change:

- user_connections Sets the maximum number of connections per user for new users. 0 means no limit is enforced. Cannot be specified with the *server-name* argument.
- db_connections Sets the maximum number of connections to the database allowed on a server. Can be specified with the *server-name* argument. This serves as the default value for new users, but does not affect existing users' settings. 0 means no limit is enforced.
- password_expiration Sets the default number of days a password is valid. 0 means the password does not expire. Can only be set globally, not per server. Cannot be specified with the *server-name* argument.
- password_warning Sets the number of days before a password expires that a warning is sent to the user console. Can only be set globally, not per server. Cannot be specified with the *server-name* argument.

value Value to which the named option is set. Values can be integers from 0 through 32767.

server-name Optional parameter allowed only when specifying db_connections. Values are the server name or all servers keyword. The latter removes server level settings and specifies the global default setting.

See also

"sp_iqaddlogin procedure" on page 741

"sp_iqmodifylogin procedure" on page 809

"LOGIN_PROCEDURE option" on page 106

Description

Note Enabling login management through sp_iqmodifylogin automatically adds existing users to the Sybase IQ Login Management tables. New users added with the GRANT CONNECT command after a database upgrade are *not* automatically added to the Sybase IQ Login Management tables.

Users added to the database with GRANT CONNECT after IQ Login Management has been enabled or a database upgrade has been done will not be managed by IQ Login Management until IQ Login Management is enabled again.

To identify such users, compare the output of SELECT * FROM SYSUSERPERM and sp_iqlistpasswordexpirations. Users who appear in the SYSUSERPERM table but not in the output of sp_iqlistpasswordexpirations are not managed by IQ Login Management. To manage these users with IQ Login Management, simply enable IQ Login Management again, as follows:

```
call sp_iqmodifyadmin('enable');
```

Errors

The following errors may occur. Causes are listed after each error.

```
Permission denied: You do not have permission to execute
the procedure sp_iqmodifyadmin.
```

Cause: A user without DBA role tried to execute sp_iqmodifyadmin.

```
RAISERROR executed: "The number of connections allowed
must specified and be greater than or equal to zero and
less than or equal to 32767."
```

Cause: A value other than 0 through 32767 was entered for user_connections.

```
RAISERROR executed: "The number of connections allowed
must specified and be greater than or equal to zero and
less than or equal to 32767."
```

Cause: A value other than 0 through 32767 was entered for db_connections.

```
RAISERROR executed: "The number of days the password is
valid must specified and be greater than or equal to
zero and less than or equal to 32767."
```

Cause: A value other than 0 through 32767 was entered for password_expiration.

```
RAISERROR executed: "The number of days to warn before
a password expires must specified and be greater than
or equal to zero and less than or equal to 32767."
```

Cause: A value other than 0 through 32767 was entered for password_warning.

```
RAISERROR executed: "Invalid input was supplied to
sp_iqmodifyadmin. Valid options are: enable, disable,
user_connections, db_connections, password_warning,
password_expiration."
```

Cause: Invalid data was entered somewhere in the command.

```
RAISERROR executed: "Server name <server name> not
found."
```

Cause: server name value specified did not match a valid server name in IQ_MPX_INFO.

```
RAISERROR executed: "Server name parameter not allowed
with this option."
```

Cause: server name value was specified for an option other than 'db_connections'

```
RAISERROR executed: "Server name parameter not allowed
with this option in non-multiplex mode."
```

Cause: procedure was called with server name argument in a nonmultiplex environment

Examples

The following procedure calls set the maximum number of connections to the database at 200. They do not enable or disable Sybase IQ Login Management.

```
call sp_iqmodifyadmin ('db_connections', 200)
```

```
sp_iqmodifyadmin 'db_connections', 200
```

If Sybase IQ Login Management is not enabled, this change is not enforced.

The following statement changes the per-server connection limit to 20 on server master:

```
sp_iqmodifyadmin('user_connections', 20, 'master'
```

sp_iqmodifylogin procedure

Function

Modifies the maximum number of connections or the password expiration interval for a given user.

Syntax1

call sp_iqmodifylogin (‘{*userid* | all overides}’, ‘*option*’, *value*['*server-name*'])

Syntax2

sp_iqmodifylogin ‘{*userid* | all overides}’, ‘*option*’, *value* [,'*server-name*'])

Permissions

DBA authority required.

Usage

userid Variable that holds the name of the account to modify.

all users If all users keyword is specified, the option and its value apply to all users except for the user DBA, which cannot be set to expire.

option Name of the option to change:

- password_expiration Password expiration interval in days, from 0 through 32767. 0 means the password does not expire. You cannot set this option on a per-server basis.
- number_of_connections Maximum number of concurrent database connections permitted for a given user. 0 means unlimited connections.

value Value to which the named option is set. Values can be from 0 through 32767.

See also

“sp_iqmodifyadmin procedure” on page 806

Description

sp_iqmodifylogin lets the DBA limit connections or set password expiration for a specified existing user, or for all other users. Changes take effect when Sybase IQ Login Management is enabled with sp_iqmodifyadmin, or immediately if Sybase IQ Login Management is already enabled.

Errors

The following errors may occur. Causes are listed after each error.

```
Permission denied: You do not have permission to change
this password.
```

Cause: A user without DBA role tried to change another user's password.

```
RAISERROR executed: "Userid <loginname> not found."
```

Cause: The user tried to change the password of a user that does not exist.

```
RAISERROR executed: "Server name <server name> not
found."
```

Cause: The server name specified did not match a valid server name in IQ_MPX_INFO.

```
RAISERROR executed: "Server name parameter not allowed
with this option in non-multiplex mode."
```

Cause: The procedure was called with the server name argument in a nonmultiplex environment.

```
RAISERROR executed: "Userid <loginname> not found."
```

Cause: The user tried to change the password of a user that does not exist.

Examples

The following stored procedure calls set password to expire in 365 days for all users except DBA.

```
sp_iqmodifylogin 'all overrides',
'password_expiration', 365
```

```
call sp_iqmodifylogin ('all overrides',
'password_expiration', 365)
```

sp_iqpassword procedure

Function

Adds or changes a password for a Sybase IQ user account.

Syntax1

call sp_iqpassword ('*caller_password*', '*new_password*' [, '*userid*'])

Syntax2

sp_iqpassword '*caller_password*', '*new_password*' [, '*userid*']

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*.

userid Login name of the user whose password is being changed by the DBA. Do not specify *userid* 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. Changes take effect when Sybase IQ Login Management is enabled with sp_iqmodifyadmin, or immediately if IQ Login Management is already enabled.

Errors

The following errors may occur. Causes are listed after each error.

```
RAISERROR executed: 'You do not have permission to
change this password.'
```

Cause: A user without DB authority tried to change another user's password.

```
RAISERROR executed: 'Userid <userid> not found.'
```

Cause: An invalid userid was specified.

```
RAISERROR executed: 'Missing procedure definition for
sp_iqrpcpassword. DBA must run sp_iqmpxdroppublication
followed by sp_iqmpxcreatepublication.'
```

Cause: Database was upgraded but post-upgrade multiplex configuration steps were not performed.

```
RAISERROR executed: 'Password change on query server not
allowed. User DBA does not have remote login permission
on write server.'
```

Cause: Database was upgraded but post-upgrade multiplex configuration steps were not performed.

Examples

The following procedure calls by a DBA set the password of the user "Jack" to jacknew.

```
sp_iqpassword 'SQL', 'jacknew', 'jack'
```

```
call sp_iqpassword ('SQL', 'jacknew', 'jack')
```

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

None required.

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 10-33: 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

See also

"sp_iqindex and sp_iqindex_alt procedures" on page 786

"sp_iqcolumn procedure" on page 751

Description

The sp_iqpkeys stored procedure displays the following information about primary keys on base and global temporary tables in a database:

Table 10-34: 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
```

sp_iqprocedure procedure

Function

Displays information about system and user-defined procedures.

Syntax

sp_iqprocedure [*proc-name*], [*proc-owner*], [*proc-type*]

Permissions

None required.

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 10-35: 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

See also

“SYSPROCEDURE system table” on page 718

CREATE PROCEDURE statement on page 485

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 10-36: 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
```

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

None required.

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_iqprocparm procedure can be invoked without any parameters. If no parameters are specified, input/output and result parameters of all the user-defined procedures (procedures not owned by dbo or SYS) are 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_iqprocparm NULL, NULL, SYSTEM` and `sp_iqprocparm NULL, user1`.

Table 10-37: 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

See also

"SYSPROCEDURE system table" on page 718

"SYSPROCPARM system table" on page 720

CREATE PROCEDURE statement on page 485

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 is displayed. 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 10-38: 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
remarks	A comment 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  remarks

sp_test    DBA         n1         normal     in         integer      4      0
(NULL)   (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  remarks

sp_iqshowcompression  dbo         @owner_name  normal     in
char         128    0      (NULL)    (NULL)
sp_iqshowcompression  dbo         @table_name  normal     in
char         128    0      (NULL)    (NULL)
sp_iqshowcompression  dbo         @column_name normal     in
char         128    0      (NULL)    (NULL)
sp_iqshowcompression  dbo         Column       result     out
char         128    0      (NULL)    (NULL)
sp_iqshowcompression  dbo         Compression  result     out
char         3      0      (NULL)    (NULL)
```

sp_iq_process_login procedure

Function: Checks that a user is permitted to connect.

Syntax: **sp_iq_process_login**

Permissions: None.

See also: "LOGIN_PROCEDURE option" on page 106

Description: When a user logs in, Sybase IQ calls the stored procedure specified by the database option LOGIN_PROCEDURE. The default setting of the LOGIN_PROCEDURE option is DBA.sp_iq_process_login.

When Sybase IQ Login Management is enabled, sp_iq_process_login checks that the user is not locked out, that the maximum number of connections for the user and database is not exceeded, and that the user's password has not expired, and then either allows user login to proceed or sends an error message. When Sybase IQ Login Management is disabled, user login proceeds without any checking.

When user login is allowed to proceed, sp_iq_process_login calls the sp_login_environment system procedure for additional processing.

This procedure is called automatically. You do not need to call it directly, unless you are creating your own login procedures. If you set LOGIN_PROCEDURE to call a different procedure, no login checking occurs.

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]*

index *index_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 *Sybase IQ System Administration Guide*.

You must specify the keywords column and index. These keywords are not case sensitive.

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.

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" in *Sybase IQ System Administration Guide*.

See also

"sp_iqindexfragmentation procedure" on page 789,"sp_iqrowdensity procedure" on page 826, and "SYSIQCOLUMN system table" on page 705

Examples

The following procedure rebuilds the default index on column *emp_lname*:

```
sp_iqrebuildindex 'employee', 'column emp_lname'
```

or

```
call sp_iqrebuildindex ('employee', 'column emp_lname')
```

The following SQL statement creates a flat default index on column *c1*:

```
CREATE TABLE mytable (c1 int IQ UNIQUE 1000000000)
```

This procedure 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')
```

sp_iqrelocate procedure

Function
Relocates specified tables and indexes on main dbspaces with relocate mode to main dbspaces with readwrite mode.

Syntax
sp_iqrelocate '*target* [**maxsize** *nMB*] [**resources** *resource-percent*]'

Parameters
target:
{database | {table *table-name* | index *index-name*} […]}

Permissions
DBA authority required.

See also
- "sp_iqdbspace procedure" on page 766, "sp_iqdbspaceinfo procedure" on page 769, and "sp_iqindexinfo procedure" on page 790
- Chapter 5, "Working with Database Objects" in the *Sybase IQ System Administration Guide*

Usage
nMB Specifies the maximum number of megabytes of data to relocate.

resource-percent Must be an integer greater than 0. The resources percentage allows you to limit the CPU utilization of the sp_iqrelocate procedure by specifying the percent of total CPUs to use.

Description
This procedure relocates specified tables and indexes on main dbspaces with relocate mode to main dbspaces with readwrite mode. If the database keyword is specified, then all objects found in relocate dbspaces are relocated. If one or more tables or indexes are specified, only the specified tables and indexes are relocated. Data that belongs to the specified tables or indexes that does not reside on relocate dbspaces is not relocated.

sp_iqrelocate can also be used to relocate tables and indexes on local dbspaces.

An object is relocated by creating a new version, the same as for DML operations. This new version must be committed before old versions are relocated and the dbspace is dropped. sp_iqrelocate does not automatically commit. You must commit the changes before they are persistent.

You can use the optional maxsize keyword to limit the amount of data relocated by the sp_iqrelocate procedure.

sp_iqrelocate returns a result set that indicates the numbers of blocks that were relocated for each object specified. The status column for each object is as follows:

- **relocated** All blocks in relocate dbspaces were relocated.
- **partial** The number of blocks in relocate dbspaces exceeds the *maxsize* parameter.
- **no relocs** The object has no blocks in relocate dbspaces.

Examples

The following command relocates the index t1c1hg on table t1 and relocates the entire table t2:

```
sp_iqrelocate 'index t1c1hg table t2';
```

ObjectName	NRelocated	RelocStatus
t1.DBA.t1c1hg	19	relocated
t2	4	relocated
t2.DBA.ASIQ_IDX_T430_C1_FP	17	relocated
t2.DBA.t2c1hng	0	no relocs

All data on dbspaces with the readwrite mode of relocate can be relocated using a single sp_iqrelocate command. The following command relocates all data on relocate dbspaces in the database:

```
sp_iqrelocate 'database';
```

ObjectName	NRelocated	RelocStatus
t1	5	relocated
t1.DBA.ASIQ_IDX_T429_C1_FP	17	relocated
t1.DBA.t1c1hg	0	no relocs
t2	0	no relocs
t2.DBA.ASIQ_IDX_T430_C1_FP	0	no relocs
t2.DBA.t2c1hng	0	no relocs

Note that the four objects with relocation status of no relocs were relocated by the previous sp_iqrelocate command.

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 10-39: 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 procedure or function
object-type not specified	The object being renamed is a table or view

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.

See also

- ALTER TABLE statement RENAME clause in Chapter 6, “SQL Statements”
- ALTER INDEX statement RENAME clause in Chapter 6, “SQL Statements”

Description

The sp_iqrename stored procedure renames user-created tables, views, columns, indexes, constraints (unique, primary key, foreign key, and check), stored procedures, 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 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
```

Renames the procedure getnamep to gettitlep:

```
sp_iqrename getnamep, gettitlep, procedure
```

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.

See also

"sp_iqcolumn procedure" on page 751

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)
```

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

This procedure is owned by dbo. Users without DBA authority must be granted execute permission for the stored procedure in order to run it.

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

This procedure 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 could be reclaimed by recreating 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 recreate, reorganize, or rebuild an index.

Example

The following procedure reports the row density on column *order_id* in table *orders*:

```
dbo.sp_iqrowdensity (‘column orders.order_id‘)
```

Table	Column	Index Type	Density
orders	order_id	Flat FP	.88

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

None required.

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.

See also

In Chapter 10, "System Procedures": "sp_iqconnection procedure" on page 753, "sp_iqcontext procedure" on page 757, and "sa_conn_info system procedure" on page 855

"CONNECTION_PROPERTY function [System]" on page 283

In Chapter 2, "Database Options": "IQGOVERN_MAX_PRIORITY option" on page 91, "IQGOVERN_PRIORITY option" on page 91, "IQGOVERN_PRIORITY_TIME option" on page 91, "MAX_QUERY_TIME option" on page 116, "QUERY_ROWS_RETURNED_LIMIT option" on page 141, "QUERY_TEMP_SPACE_LIMIT option" on page 142, "MAX_CURSOR_COUNT option" on page 113, and "MAX_STATEMENT_COUNT option" on page 116

"AppInfo connection parameter [App]" in Chapter 4, "Connection and Communication Parameters" in the *Sybase IQ System Administration Guide*

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 10-40: sp_iqshowpsexe columns

Column name	Description
connection_id	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\\
                ASIQ-12_7\\win32\\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
```

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** | **local** | [**table** *table-name* | **index** *index-name*] [...] ']

Permissions

DBA authority required.

See also

- "sp_iqindexinfo procedure" on page 790, "sp_iqdbspace procedure" on page 766, and "sp_iqdbspaceinfo procedure" on page 769
- Chapter 5, "Working with Database Objects" in the *Sybase IQ System Administration Guide*

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. If the database is a multiplex database, the default is main, the size of the shared IQ main store. The optional parameter local specifies only information about the local IQ store owned by the query server.

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 asiqdemo database. Lines of output for some tables and indexes have been removed in this example.

```
             Name                                  NBlocks    dbspace_name
contact                                            19         IQ_SYSTEM_MAIN
sales_order_items.DBA.ASIQ_IDX_T205_C5_FP          56         IQ_SYSTEM_MAIN
contact.DBA.ASIQ_IDX_T206_C10_FP                   55         IQ_SYSTEM_MAIN
contact.DBA.ASIQ_IDX_T206_C1_FP                    61         IQ_SYSTEM_MAIN
...
contact.DBA.ASIQ_IDX_T206_C9_FP                    55         IQ_SYSTEM_MAIN
contact.DBA.ASIQ_IDX_T206_I11_HG                   19         IQ_SYSTEM_MAIN
customer                                           20         IQ_SYSTEM_MAIN
customer.DBA.ASIQ_IDX_T207_C1_FP                   61         IQ_SYSTEM_MAIN
customer.DBA.ASIQ_IDX_T207_C2_FP                   55         IQ_SYSTEM_MAIN
...
customer.DBA.ASIQ_IDX_T207_I10_HG                  19         IQ_SYSTEM_MAIN
...
```

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)
```

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. In a multiplex, this procedure applies to the server on which it runs. If a query server has no IQ Local Store, it returns 0 in the first two out parameters.

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 10-41: sp_iqspaceused columns

Column name	Description
mainKB	The total Main IQ Store space in kilobytes.
mainKBUsed	The number of kilobytes of Main IQ Store space used by the database.
tempKB	The total Temporary IQ Store space in kilobytes.
tempKBUsed	The number of kilobytes of Temporary IQ 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_iqstatus procedure

Function

Displays a variety of Sybase IQ status information about the current database.

Syntax

sp_iqstatus

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. On a query server in a multiplex, this procedure lists information about the IQ Local Store as well as the shared IQ Store and IQ Temporary Store.

If 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. See “sp_iqversionuse procedure” on page 846.

The system stored procedure sp_iqspaceused returns a subset of the same information as that 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 830.

Example

The following output is from the sp_iqstatus stored procedure:

```
Adaptive Server IQ (TM)              Copyright (c) 1992-2006 by Sybase, Inc.
                                      All rights reserved.
Version:                             12.7.0/040810/P/GA/MS/
                                     Windows 2000/32bit/2006-06-10
09:54:19
Time Now:                            2006-06-11 18:53:34.274
Build Time:                          2006-06-10 09:54:19
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 DB Spaces:                 1
Number of Temp Spaces:               1
DB Blocks: 1-5632                    IQ_SYSTEM_MAIN
Temp Blocks: 1-2816                  IQ_SYSTEM_TEMP
Create Time:                         2006-06-03 14:14:06.124
Update Time:                         2006-06-03 14:14:26.687
Main IQ Buffers:                     127, 16Mb
Temporary IQ Buffers:                95, 12Mb
Main IQ Blocks Used:                 4541 of 5632, 80%=35Mb, Max Block#:
5120
Temporary IQ Blocks Used:            65 of 2816, 2%=0Mb, Max Block#: 0
Main Reserved Blocks Available:      512 of 512, 100%=4Mb
Temporary Reserved Blocks Available: 256 of 256, 100%=2Mb
IQ Dynamic Memory:                   Current: 41mb, Max: 41mb
Main IQ Buffers:                     Used: 4, Locked: 0
Temporary IQ Buffers:                Used: 4, Locked: 0
Main IQ I/O:                         I: L168/P2 O: C2/D16/P15 D:0 C:100.0
Temporary IQ I/O:                    I: L862/P0 O: C136/D150/P17 D:132
C:100.0
Other Versions:                      0 = 0Mb
Active Txn Versions:                 0 = C:0Mb/D:0Mb
```

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

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

None required.

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 10-42. Without the prefix, both stores are monitored. For example, if you specify “mbufman”, only the Main IQ Store buffer manager is monitored. If you specify “mbufman tbufnam” or “bufman”, both the Main and Temporary Store buffer managers are monitored.

Table 10-42: sp_iqsysmon report section abbreviations

Report section or IQ component	Abbreviation
Buffer manager	(m/t)bufnam
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 10-42 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 10-43: 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

See also

IQ UTILITIES statement on page 576

Chapter 6, “Monitoring and Tuning Performance” in the *Sybase IQ Performance and Tuning Guide*

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 asiqdemo, writes results in the file *asiqdemo.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 %)
```

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'*]

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 10-44 lists the allowed values for the *table_type* parameter:

Table 10-44: 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 10-45: sp_iqtable columns

Column name	Description
table_name	The name of the table.
table_type	SYSTEM, BASE – a base table VIEW – a view TEMP, JVT – all join virtual tables of both IQ Store and IQ Local Stores, ALL GBL – a global temporary table JVT_MAIN – join virtual table of the IQ Store JVT_LOCAL – join virtual table of an IQ Local Store.
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 Temp Store MAIN – IQ Store LOCAL – IQ Local Store SYSTEM – Catalog Store
remarks	User comments added with the COMMENT statement.

Examples

The following variations in syntax both return information about the table department:

```
call sp_iqtable ('department')
sp_iqtable table_name='department'
```

table_name	table_type	table_owner	server_type	location	remarks
department	BASE	DBA	IQ	MAIN	(NULL)

The following variations in syntax both return all tables that are owned by table owner DBA:

```
sp_iqtable NULL,DBA
sp_iqtable table_owner='DBA'
```

table_name	table_type	table_owner	server_type	location	remarks
contact	BASE	DBA	IQ	MAIN	(NULL)
customer	BASE	DBA	IQ	MAIN	(NULL)
department	BASE	DBA	IQ	MAIN	(NULL)
employee	BASE	DBA	IQ	MAIN	(NULL)
fin_code	BASE	DBA	IQ	MAIN	(NULL)

table_name	table_type	table_owner	server_type	location	remarks
fin_data	BASE	DBA	IQ	MAIN	(NULL)
product	BASE	DBA	IQ	MAIN	(NULL)
sales_order	BASE	DBA	IQ	MAIN	(NULL)
sales_order_items	BASE	DBA	IQ	MAIN	(NULL)

sp_iqtablesize procedure

Function

Returns the size of the specified table.

Syntax

sp_iqtablesize (*table_owner.table_name*)

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 10-46: 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.tab1')
```

Ownername	Tablename	Columns	KBytes	Pages	CompressedPages	NBlocks
DBA	tab1	16	5838548	124260	91344	1459637

sp_iqtransaction procedure

Function

Shows information about transactions and versions.

Syntax

sp_iqtransaction

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.

The 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 10-47: sp_iqtransaction columns

Column name	Description
Name	The name of the server.
Userid	The user ID for the connection.

Column name	Description
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 nonmultiplex databases and multiplex write servers, the VersionID is the same as the TxnID. In multiplex query servers, the VersionID is the TxnID of the transaction that created the database version on the multiplex write server. 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.
MainTableKBCreated	The number of kilobytes of IQ Store space created by this transaction.
MainTableKBDropped	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.
TempTableKBCreated	The number of kilobytes of IQ Temporary Store space created by this transaction for storage of IQ temporary table data.
TempTableKBDropped	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.

Column name	Description
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.
Dbremote	A bit data column that indicates the transaction is an internal transaction used to replicate multiplex version information between a query server and the write server within a multiplex database.
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.

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

MainTableKBCreated MainTableKBDropped TempTableKBCreated TempTableKBDropped
================== ================== ================== ==================
                 0                  0              65824                  0
                 0                  0                  0                  0
                 0                  0                  0                  0
                 0                  0                  0                  0
              3960                152                  0                  0
                 0                  0                  0                  0
              2440               1992                  0                  0
                 0                  0                  0                  0
                 0                  0               2912              22096
                 0                  0                  0                  0

TempWorkSpaceKB TxnCreateTime      Dbremote CursorCount SpCount SpNumber
=============== ================== ======== =========== ======= =======
              0    2006-06-26 13:17:27.612        0           1       3       2
         102592    2006-06-26 13:27:28.491        0           1       1       0
              0    2006-06-26 13:30:27.548        0           0       1       0
              0    2006-06-26 13:31:27.151        0           0      24     262
              0    2006-06-26 13:35:02.128        0           0       0       0
              0    2006-06-26 13:43:58.805        0           0      39     408
            128    2006-06-26 13:45:28.379        0           0       1       0
              0    2006-06-26 14:05:15.759        0           0      42     413
            680    2006-06-26 14:57:51.104        0           1       2      20
              0    2006-06-26 15:09:30.319        0           0       1       0
```

sp_iqversionuse procedure

Function

Displays version usage for the IQ Main store.

Syntax

call dbo.sp_iqversionuse ( )

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.

The procedure produces a row for each user of a version. Run sp_iqversionuse first on the write server to determine which versions should be released and the amount of space in KB to be released when the version is no longer in use. Connection IDs are displayed in the IQConn column for users connected to the write server. Version usage due to query servers is displayed as the query server name with connection ID 0.

Run sp_iqversionuse on multiplex query servers to determine individual connections to query servers. Users from other servers are not displayed on a query server.

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 query server to the write server. WasReported is 0 initially on a write server for new versions. WasReported changes to 1 once SQL Remote replicates version usage information back to the write server. If WasReported is 0 for an extended period, SQL Remote might be stopped.

Table 10-48: 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

In this example, the oldest version 42648 is in use by connection 108 on the write server (*mpxw*). Committing or rolling back the transaction on connection 108 releases 7.9MB of space. Version 42686 is in use by query server (*mpxq*) according to output from the write server. Using the query server output, the actual connection is connection 31. The actual amount of space returned from releasing version 42686 depends on whether 42648 is released first.

WasReported is 0 for versions 42715 and 42728 on the write server because these are new versions that have not yet been replicated by SQL Remote. Since version 42728 does not appear on the query server output, it has not yet been used by the query server.

The following output is returned when sp_iqversionuse executes on the write server *mpxw*:

```
call dbo.sp_iqversionuse
```

VersionID	Server	IQConn	WasReported	MinKBRelease	MaxKBRelease
42648	'mpxw'	108	1	7920	7920
42686	'mpxq'	0	1	7920	304
42702	'mpxq'	0	1	0	688
42715	‘mpxq'	0	0	0	688
42728	'mpxq'	0	0	0	688

The following output is returned when sp_iqversionuse executes on the query server (*mpxq*):

```
call dbo.sp_iqversionuse
```

VersionID	Server	IQConn	WasReported	MinKBRelease	MaxKBRelease
42686	'mpxq'	31	1	0	0
42715	'mpxq'	00	1	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'*]

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 10-49 lists the allowed values for the *view_type* parameter.

Table 10-49: 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 10-50: 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

Examples

The following variations in syntax both return information about the view deptview:

```
call sp_iqview('deptview')
sp_iqview view_name='deptview'
```

view_name	view_owner	view_def	remarks
deptview	DBA	create view DBA.deptview(vdep	(NULL)

The following variations in syntax both return all views that are owned by view owner DBA:

```
sp_iqview NULL,DBA
sp_iqview view_owner='DBA'
```

view_name	view_owner	view_def	remarks
deptview	DBA	create view DBA.deptview(vdep	(NULL)
empview	DBA	create view DBA.empview(vemp	(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

None required.

Description

The sp_iqwho stored procedure displays information about all current users and connections, or about a particular user or connection.

Table 10-51: 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.
TmpTblSpaceKB	The size of temporary table space in kilobytes; 0 if no temporary table space is used
TmpWrkSpaceKB	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 10-52 maps the Adaptive Server Enterprise sp_who columns to the columns displayed by sp_iqwho.

Table 10-52: 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; can be confusing in a multiplex environment, so currently is 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 10-53: 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"`

See also

"sp_iqconnection procedure" on page 753

"sa_conn_info system procedure" on page 855

Example

Display all active connections:

```
ConnHandle IQConnID Userid ReqType      IQCmdType            BlockedOn BlockUserid IQCursors
IQThreads IQIdle SAIdle TmpTblSpaceKB TmpWrkSpaceKB

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
```

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.

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: sa_checkpoint_execute 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 in 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 non-zero 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 in conjunction with disk mirroring to split mirrored devices.

Example

The following statement issues a checkpoint, copies all the asiqdemo database files to a backup directory, then completes the checkpoint.

```
sa_checkpoint_execute 'cp asiqdemo.* /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

None

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 the *Adaptive Server Anywhere Database Administration Guide*.

For information about the -zl command line option, see Chapter 1, “Running the Database Server” in *Sybase IQ Utility Guide.*

For information about the remember_last_statement setting, see “sa_server_option system procedure” on page 865.

sa_conn_info system procedure

Function

Reports connection property information.

Syntax

sa_conn_info ([*connection-id*])

Permissions

None.

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
- ProcessTime
- Port
- ReqType
- CommLink
- NodeAddr
- LastIdle
- CurrTaskSwitch
- BlockedOn
- UncommitOp

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*])

Description
Returns the connection ID as Number, and the 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 the section "Database properties" in the chapter "Database Performance and Connection Properties" in the *Adaptive Server Anywhere Database Administration Guide*.

Example

```
sa_conn_properties
```

```
569851433,29,'CacheHits','Cache hits','0'
569851433,30,'CacheRead','Cache reads','0'
569851433,31,'DiskRead','Disk reads','57'
569851433,32,'DiskSyncRead','Disk synchronous reads','0'
569851433,33,'DiskWaitRead','Disk wait for reads','0'
569851433,34,'DiskWaitWrite','Disk wait for writes','0'
569851433,35,'CacheReadTable','Cache table reads','0'
569851433,36,'CacheReadIndLeaf','Cache index leaf reads','0'
569851433,37,'CacheReadIndInt','Cache index interior reads','0'
569851433,38,'DiskReadTable','Disk table reads','0'
569851433,39,'DiskReadIndLeaf','Disk index leaf reads','0'
569851433,40,'DiskReadIndInt','Disk index interior reads','0'
569851433,41,'CacheWrite','Cache writes','0'
569851433,42,'DiskWrite','Disk writes','4'
```

Note CacheHits is always reported as 0 by sa_conn_properties, as this information is not stored by user connection. 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_conn_properties_by_conn system procedure

Function
Reports connection property information.

Syntax
sa_conn_properties_by_conn ([*property-name*])

Permissions
None.

See also
"sa_conn_properties system procedure" on page 856

Description
This is a variant on the sa_conn_properties system procedure. It returns results only for connection properties that match the *property-name* string. You can use wildcards in *property-name*, as the comparison uses a LIKE operator. The result set is sorted by connection number and property name.

For a listing of available connection properties, see the section "Database properties" in the chapter "Database Performance and Connection Properties" in the *Adaptive Server Anywhere Database Administration Guide*.

Examples

- The following statement returns CacheHits settings:

```
sa_conn_properties_by_conn CacheHits

569851433,29,'CacheHits','Cache hits','0'
```

- The following statement returns the AnsiNull option setting for the current connection:

```
call sa_conn_properties_by_conn( 'ansinull' )

569851433,202,'Ansinull','Ansinull','ON'
```

- The following statement returns the ANSI-related option settings for the current connection:

```
call sa_conn_properties_by_conn( 'ansi%' )

'569851433,198,'Ansi_blanks',
'Ansi_blanks','OFF'
569851433,225,
'Ansi_close_cursors_on_rollback',
'Ansi_close_cursors_on_rollback','ON'
569851433,199,'Ansi_integer_overflow',
'Ansi_integer_overflow','OFF'
569851433,203,'Ansi_permissions',
'Ansi_permissions','ON'
569851433,202,'Ansinull','Ansinull','ON'
```

sa_conn_properties_by_name system procedure

Function
Reports connection property information.

Syntax
sa_conn_properties_by_name ([*connection-id*])

Permissions
None.

See also
"sa_conn_properties system procedure" on page 856

Description

This is a variant on the sa_conn_properties system procedure. It returns the same result columns. The information is sorted by property name and connection number.

For a listing of available connection properties, see the section "Database properties" in the chapter "Database Performance and Connection Properties" in the *Adaptive Server Anywhere Database Administration Guide*.

Example

```
sa_conn_properties_by_name
```

For an example of the results returned by sa_conn_properties_by_name, see "sa_conn_properties system procedure" on page 856.

sa_db_info system procedure

Function

Reports database property information.

Syntax

sa_db_info ([*database-id*])

Permissions

None.

See also

"sa_db_properties system procedure" on page 859

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 10-54 lists sample values.

```
sa_db_info
```

```
0,'asiqdemo','
/sys1/users/janed/sybase/ASIQ-12_7/demo/
asiqdemo.db',
1,4096,'/sys1/users/janed/sybase/ASIQ-12_7/demo/
asiqdemo.log'
```

Table 10-54: sa_db_info sample values

Property	Value
Number	0
Alias	asiqdemo
File	c:\ASIQ-12_7\demo\asiqdemo.db
ConnCount	1
PageSize	4096
LogName	c:\ASIQ-12_7\demo\asiqdemo.log

sa_db_properties system procedure

Function: Reports database property information.

Syntax: **sa_db_properties** ([*database-id*])

Permissions: None.

See also: "sa_db_info system procedure" on page 858

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
```

```
0,125,'Name','Database name','asiqdemo'
0,126,'Alias','Mounted database name','asiqdemo'
0,127,'File','Database file',
'/sys1/users/janed/sybase/ASIQ-12_7/demo/asiqdemo.db'
0,128,'PageSize','Database page size','4096'
0,129,'LogName','Database log file name',
'/sys1/users/janed/sybase/ASIQ-12_7/demo/
asiqdemo.log'0,131,'ConnCount','Number of connections','1'
0,146,'FileVersion',
'Database file version number','1005'
0,147,'CheckpointUrgency',
'Database checkpoint urgency','30'
0,148,'RecoveryUrgency',
'Database recovery urgency','0'
0,151,'IQStore','IQ store is on/off','ON'
0,163,'CharSet','Character Set','iso_1'
0,164,'MultiByteCharSet',
'Multi Byte Character Set ( on/off )','OFF'
0,165,'Language','Language','unknown'
```

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 containing one or more of:

Permissions: DBA authority required.

See also: "AUDITING option [database]" on page 45

Description

sa_enable_auditing_type works in conjunction 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, you must 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')
```

sa_eng_properties system procedure

Function

Reports database server property information.

Syntax

sa_eng_properties

Permissions

None.

Description

Returns the PropNum, PropName, PropDescription, and Value for each available server property.

For a listing of available engine properties, see the section “Database properties” in the chapter “Database Performance and Connection Properties” in the *Adaptive Server Anywhere Database Administration Guide*.

Examples

- The following statement returns a set of available server properties:

```
call sa_eng_properties()
```

PropNum	PropName	...
0	IdleCheck	...
1	IdleWrite	...
2	IdleChkPt	...
...	...	...

- The following statement returns a set of available server properties:

```
sa_eng_properties

0,'IdleCheck','Idle I/O checked','0'
1,'IdleWrite','Idle I/O writes','0'
2,'IdleChkpt','Idle I/O checkpoints','0'
3,'IdleChkTime','Idle I/O checkpoint time','0'
4,'Chkpt','Checkpoints','5'
5,'ChkptPage','Checkpoint log pages','19'
6,'ChkptFlush','Checkpoint flushed pages','24'
7,'ExtendDB','Extend database file writes','0'
8,'ExtendTempWrite','Extend temporary file writes','198'
9,'FreeWritePush','Free list write to pushable list','0'
10,'FreeWriteCurr','Free list write to current list','0'
11,'CommitFile','Commit writes to disk','59'
12,'PendingReq','Pending requests detected','0'
13,'CurrRead','Active disk read requests','0'
14,'MaxRead','Maximum active disk read requests','3'
15,'CurrWrite','Active disk write requests','0'
16,'MaxWrite','Maximum active disk write requests','4'
17,'CurrIO','Active disk read/write requests','0'
18,'MaxIO','Maximum active disk read/write requests','4'
19,'JavaNSSize','Java VM Namespace size','0'
20,'IOToRecover','','0'
```

sa_table_page_usage system procedure

Function Reports information about the usage of database tables.

Syntax **sa_table_page_usage**

Description The results include the same information provided by the Information utility.

For information on the Information utility, see 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.

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.

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.

See also: "sa_db_info system procedure" on page 858

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 10-55 lists the meaning of the sa_make_object parameters.

Table 10-55: 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' )
  ```

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.

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 zip codes.
- 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)
```

Permissions

None

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 zip code in the range.

```
SELECT row_num AS r1, row_num+9999  AS r2,
count(zip_code) AS zips_in_range FROM
sa_rowgenerator(0,99999,10000) D LEFT JOIN employee ON
zip_code 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 emp(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.

See also

"sa_get_request_profile system procedure," "sa_get_request_times system procedure," and "sa_statement_text system procedure" in *Adaptive Server Anywhere SQL Reference*.

Description

Database administrators can use this procedure to override some database server options without restarting the database server.

The following options can be reset:

Option name	Values	Default
Disable_connections	ON or OFF	OFF
Liveness_timeout	Integer, in seconds	120
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.

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 more information, see -tl command line option in "Server command-line switches" on page 8 in Chapter 1, "Running the Database Server" in the *Sybase IQ Utility 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 the *Adaptive Server Anywhere SQL Reference*.

For more information about viewing procedure profiling information in Sybase Central, see "Profiling database procedures" in the *Sybase IQ 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, see the -tq server option in Chapter 1, "Running the Database Server" in the *Sybase IQ 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 *Sybase IQ 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 Can be ALL, SQL, NONE, or SQL+hostvars. 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. 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 *Sybase IQ Utility Guide*. See "-zr level" on page 29 in the *Sybase IQ Utility Guide* for a list of requests that are logged by SQL request-level logging.

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 )
```

Example

The following statement disallows new connections to the database server.

```
call sa_server_option( 'disable_connections', 'ON')
```

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: None.

See also: "sa_set_http_option system procedure" on page 870

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.

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: None.

See also: "sa_set_http_header system procedure" on page 870

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.

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.

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 *Sybase IQ Utility Guide*.

Table 10-56 lists the meaning of the sa_validate parameters.

Table 10-56: 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.

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.

Permissions — None

Side effects — None

sp_login_environment system procedure

Function — Sets connection options when users log in.

Permissions — None.

Syntax — **sp_login_environment**

See also — "LOGIN_PROCEDURE option" on page 106

Description — At start-up, sp_login_environment is called by DBA.sp_iq_process_login, the default procedure called by the LOGIN_PROCEDURE database option.

Sybase recommends that you *not* edit this procedure. Instead, to change the login environment, set the LOGIN_PROCEDURE option to point to a different procedure.

For more information about setting the LOGIN_PROCEDURE option to the name of a new procedure, see Chapter 15, "Sybase IQ as a Data Server" in the *Sybase IQ System Administration Guide*.

Here is the text of sp_login_environment:

```
CREATE PROCEDURE dbo.sp_login_environment()
BEGIN
  IF connection_property('CommProtocol')='TDS' THEN
    CALL dbo.sp_tsql_environment()
  END IF
END
```

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

None.

See also

Chapter 17, "Server Classes for Remote Data Access" and Chapter 16, "Accessing Remote Data" in the *Sybase IQ System Administration Guide*

CREATE SERVER statement on page 494

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.

Standards and compatibility

- **Sybase** Supported by Open Client/Open Server.

Example

Gets a list of the columns in the sysobjects table in the production database in an ASE server named asetest:

```
sp_remote_columns asetest, sysobjects,
null, production
```

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

None.

See also

Chapter 16, "Accessing Remote Data" and Chapter 17, "Server Classes for Remote Data Access" in the *Sybase IQ System Administration Guide*

CREATE SERVER statement on page 494

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 10-57 lists the sp_remote_exported_keys parameters.

Table 10-57: 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')
```

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

None.

See also

Chapter 16, “Accessing Remote Data” and Chapter 17, “Server Classes for Remote Data Access” in the *Sybase IQ System Administration Guide*.

CREATE SERVER statement on page 494

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 10-58 lists the sp_remote_imported_keys parameters.

Table 10-58: 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')
```

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: None

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: None.

See also: CREATE SERVER statement on page 494

Chapter 16, “Accessing Remote Data” and Chapter 17, “Server Classes for Remote Data Access” in the *Sybase IQ System Administration Guide*

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 Adaptive Server 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."

Standards and compatibility

- **Sybase** Supported by Open Client/Open Server.

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
```

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

None.

See also

CREATE SERVER statement on page 494

Chapter 16, "Accessing Remote Data" and Chapter 17, "Server Classes for Remote Data Access" in the *Sybase IQ System Administration Guide*

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.

Standards and compatibility

- **Sybase** Supported by Open Client/Open Server.

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'
```

sp_tsql_environment system procedure

Function

To set connection options when users connect from jConnect or Open Client applications.

Syntax

sp_tsql_environment

Permissions

None.

See also

"sp_login_environment system procedure" on page 872

"LOGIN_PROCEDURE option" on page 106

Description

At startup, sp_login_environment is called by DBA.sp_iq_process_login, the default procedure called by the LOGIN_PROCEDURE database option. If the connection uses the TDS communication protocol (that is, if it is an Open Client or jConnect 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 15, "Sybase IQ as a Data Server" in the *Sybase IQ System Administration Guide*.

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 AUTOMATIC_TIMESTAMP='ON'
  end if;
  set temporary option ANSINULL='OFF';
  set temporary option TSQL_VARIABLES='ON';
```

```
  set temporary option ANSI_BLANKS='ON';
  set temporary option TSQL_HEX_CONSTANT='ON';
  set temporary option CHAINED='OFF';
  set temporary option QUOTED_IDENTIFIER='OFF';
  set temporary option ALLOW_NULLS_BY_DEFAULT='OFF';
  set temporary option CONTINUE_AFTER_RAISERROR='ON';
  set temporary option FLOAT_AS_DOUBLE='ON';
  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
```

Multiplex system procedures

The procedures in this section affect multiplex databases and servers. These procedures are intended to be called by a program. Do not run them by specifying the procedure name in an ISQL window. Generally, these procedures are intended for use:

- Within the server
- From Sybase Central
- From administration scripts

Most of these procedures require DBA privileges.

sp_iqmpxcountdbremote procedure

Function

Returns a count of dbremote connections for a multiplex database. This is a function implemented as a stored procedure.

Syntax

dbo.sp_iqmpxcountdbremote ( )

Examples

When dbremote is running, the result is 5 connections. The following ISQL statement returns results as a table:

```
select dbo.sp_iqmpxcountdbremote()
```

dbo.sp_iqmpxcountdbremote(*)

5

You can also display the output as a table using the following SQL:

```
begin
    declare dbremotes int;
    set dbremotes = dbo.sp_iqmpxcountdbremote();
    select dbremotes;
end
```

sp_iqmpxgetconnversion procedure

Function

Displays the version number for a specified connection.

Syntax

call sp_iqmpxgetconnversion ( )

Usage

On a multiplex query server, returns in an UNSIGNED BIGINT the version number that the current connection uses for the current transaction. On a write server, always returns 0.

The version number is a database-wide monotonically-increasing integer that increases any time the write server commits new data in the IQ Main store.

Description

This procedure lets you determine versions across connections on one or more multiplex query servers in a multiplex environment.

For instance, assume that two different connections are made to query servers at nearly the same time, and you need to coordinate by SELECT statements between the two connections so that they are using the same version of the data. If any transaction committed between the moment of the first and the second connections, they see two different versions.

This procedure lets you determine whether both connections are using the same version of the data.

You cannot determine whether write server connections see the same version of the data.

sp_iqmpxreplacewriteserver procedure

Function

Converts the query server on which it runs into the new write server for the multiplex. Must be called on the query server. Other steps are needed to

move to a new write server. For details, see Chapter 5, “Working with Database Objects” in the *Sybase IQ System Administration Guide*.

Syntax

call sp_iqmpxreplacewriteserver(‘servername’)

Description

Drops and recreates main IQ Store definitions for the new write server to match those of the query server. Drops any IQ Temporary Store definitions for the former write server. Adjusts SYSIQFILE, IQ_MPX_STATUS, IQ_MPX_INFO, and SQL Remote configuration.

Table 10-59: sp_iqmpxreplacewriteserver columns

Column name	Data type	Description
servername	varchar(30)	Name for the new write server. Must differ from each server name currently in the multiplex.

Usage

Must run on query server.

Permissions

Must have DBA authority.

sp_iqmpxvalidate procedure

Function

Checks multiplex configuration for inconsistencies.

Syntax

call dbo.sp_iqmpxvalidate(‘ show_msgs’)

Description

Multiple checks on tables SYS.SYSIQFILE and DBA.IQ_MPX_INFO, and SQL Remote configuration. May run on any server. Returns a result to the caller: severity. Values are:

Value	Description
0	No configuration errors
1	Dynamic state is not as expected; for example, dbremote process not running
2	Nonfatal configuration error; for example, multiplex operation impaired
3	Fatal configuration problem; for example, one or more servers might not start

If called interactively, the stored procedure also returns a table of the errors found, if any, unless the calling parameter is not 'Y'.

Each error indicates its severity. If there are no errors, the procedure returns “No errors detected”.

sp_iqmpxversioninfo procedure

Function

Shows the current version information for this server. Information includes server type (write server, query server, single-node mode) and synchronization status.

Syntax

sp_iqmpxversioninfo( )

Description

Table 10-60: sp_iqmpxversioninfo columns returned

Column	Data type	Description
CatalogID	unsigned bigint	Catalog version on this server
VersionID	unsigned bigint	Latest version available on this server
OAVID	unsigned bigint	Oldest active version on this server
ServerType	char(1)	Type of server: 'S,' 'W,' or 'Q'
CatalogSync	char(1)	In catalog synchronization? 'T' or 'F'
WCatalogID	unsigned bigint	Catalog version on the write server
WVersionID	unsigned bigint	Latest version available on the write server

sp_mpxcfg_<*servername*> procedure

Function

Sets up query server named *servername* for SQL Remote replication.

Syntax

call "DBA".sp_mpxcfg_<*servername*>(' ')

Description

Sybase IQ calls this procedure when synchronizing query servers. This procedure in turn runs specified procedure or procedures on the named query server. When finished, this procedure returns the following message in the server log: `Query server auto-configuration complete`. If the query server is already configured or if you run sp_mpxcfg_*servername* on a write server, the procedure does nothing.

Permissions

Must have DBA authority.

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'
```

You might need to reset the width of your DBISQL output to see the full text, by selecting Command→Options and entering a new Limit Display Columns value.

Adaptive Server Enterprise system procedures

Table 10-61 describes the Adaptive Server Enterprise system procedures provided in Sybase IQ.

Table 10-61: 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 SYSMESSAGES and 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 10-62. Sybase IQ also has similar customized stored procedures for some of these ASE catalog procedures.

Table 10-62: 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

CHAPTER 11

System Views

About this chapter

This chapter lists predefined views for the Sybase IQ system tables.

The system tables use numbers to identify tables, user IDs, and so forth. Although this is efficient for internal use, it makes these tables difficult for people to interpret. A number of predefined system views are provided that present the information in the system tables in a more 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.

Contents

SYSARTICLECOLS system view

```
CREATE VIEW SYS.SYSARTICLECOLS
AS SELECT (select publication_name FROM
    SYS.SYSPUBLICATION AS p
WHERE p.publication_id=ac.publication_id) AS
publication_name,
       (select table_name FROM SYS.SYSTABLE AS t
       WHERE t.table_id=ac.table_id) AS table_name,
       select column_name FROM SYS.SYSCOLUMN AS c
       WHERE c.table_id=ac.table_id
    AND c.column_id=ac.column_id) AS column_name
FROM SYS.SYSARTICLECOL AS ac
```

Presents a readable version of the table SYSARTICLECOLS.

SYSARTICLES system view

```
CREATE VIEW SYS.SYSARTICLES
  AS SELECT(select publication_name FROM
       SYS.SYSPUBLICATION AS p
    WHERE p.publication_id=a.publication_id) AS
       publication_name,
    (select table_name FROM SYS.SYSTABLE AS t
    WHERE t.table_id=a.table_id) AS table_name,
       where_expr,subscribe_by_expr
FROM SYS.SYSARTICLE AS a
```

Presents a readable version of the table SYSARTICLES.

SYSCAPABILITIES system view

```
CREATE VIEW SYS.SYSCAPABILITIES
AS
SELECT t1.capid,srvid,capname,capvalue
FROM
    SYS.SYSCAPABILITY as t1
    JOIN SYS.SYSCAPABILITYNAME as t2
    ON t1.capid = t2.capid
```

Presents the data from the system tables SYSCAPABILITY and SYSCAPABILITYNAME in a readable format.

SYSCATALOG system view

```
CREATE VIEW SYS.SYSCATALOG ( creator,
    tname, dbspacename, tabletype, ncols,
    primary_key, "check", remarks )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
              WHERE user_id = SYSTABLE.creator ),
       table_name,
       ( SELECT dbspace_name from SYS.SYSFILE
              WHERE file_id = SYSTABLE.file_id ),
       IF table_type='BASE' THEN 'TABLE'
          ELSE table_type ENDIF,
       ( SELECT count(*) FROM SYS.SYSCOLUMN
          WHERE table_id = SYSTABLE.table_id ),
       IF primary_root = 0 THEN 'N' ELSE 'Y' ENDIF,
       IF table_type <> VIEW' THEN view_def ENDIF,
       remarks
FROM SYS.SYSTABLE
```

Lists all the tables and views from SYSTABLE in a readable format.

SYSCOLAUTH system view

```
CREATE VIEW SYS.SYSCOLAUTH ( grantor, grantee,
    creator, tname, colname )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
          WHERE user_id = SYSCOLPERM.grantor ),
       ( SELECT user_name FROM SYS.SYSUSERPERM
          WHERE user_id = SYSCOLPERM.grantee ),
       ( SELECT user_name
          FROM SYS.SYSUSERPERM == SYS.SYSTABLE
          WHERE table_id = SYSCOLPERM.table_id ),
       ( SELECT table_name FROM SYS.SYSTABLE
          WHERE table_id = SYSCOLPERM.table_id ),
       ( SELECT column_name FROM SYS.SYSCOLUMN
```

```
          WHERE table_id = SYSCOLPERM.table_id
          AND column_id = SYSCOLPERM.column_id )
FROM SYS.SYSCOLPERM
```

Presents column update permission information in SYSCOLPERM in a readable format.

SYSCOLUMNS system view

```
CREATE VIEW SYS.SYSCOLUMNS ( creator, cname, tname,
coltype, nulls, length, syslength,
in_primary_key, "colno", default_value, remarks )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
          WHERE user_id = SYSTABLE.creator ),
       column_name, table_name,
       ( SELECT domain_name FROM SYS.SYSDOMAIN
          WHERE domain_id = SYSCOLUMN.domain_id ),
       nulls, width, scale, pkey, column_id,
       "default", SYSCOLUMN.remarks
FROM SYS.SYSCOLUMN == SYS.SYSTABLE
```

Presents a readable version of the table SYSCOLUMN.

Note The “S” at the end of the view name distinguishes it from the SYSCOLUMN table.

SYSFOREIGNKEYS system view

```
CREATE VIEW SYS.SYSFOREIGNKEYS ( foreign_creator,
    foreign_tname, primary_creator,
    primary_tname, role, columns )
AS
SELECT ( SELECT user_name FROM
          SYS.SYSUSERPERM == SYS.SYSTABLE
          WHERE table_id = foreign_table_id ),
       ( SELECT table_name FROM SYS.SYSTABLE
          WHERE table_id = foreign_table_id ),
```

```
            ( SELECT user_name
               FROM SYS.SYSUSERPERM == SYS.SYSTABLE
               WHERE table_id = primary_table_id ),
            ( SELECT table_name FROM SYS.SYSTABLE
               WHERE table_id = primary_table_id ), role,
            ( SELECT list( string( FK.column_name,
               ' IS ', PK.column_name ) )
               FROM SYS.SYSFKCOL KEY JOIN
               SYS.SYSCOLUMN FK, SYS.SYSCOLUMN PK
               WHERE foreign_table_id =
               SYSFOREIGNKEY.foreign_table_id
               AND foreign_key_id =
                      SYSFOREIGNKEY.foreign_key_id
               AND PK.table_id =
                      SYSFOREIGNKEY.primary_table_id
               AND PK.column_id =
                      SYSFKCOL.primary_column_id )
FROM SYS.SYSFOREIGNKEY
```

Presents foreign-key information from SYSFOREIGNKEY and SYSFKCOL in a readable format.

SYSGROUPS system view

```
CREATE VIEW SYS.SYSGROUPS ( group_name, member_name )
AS
SELECT g.user_name, u.user_name
FROMSYS.SYSGROUP,
       SYS.SYSUSERPERM g,
       SYS.SYSUSERPERM u
WHERE group_id = g.user_id
AND group_member = u.user_id
```

Presents group information from SYSGROUP in a readable format.

SYSINDEXES system view

```
CREATE VIEW SYS.SYSINDEXES ( icreator, iname, fname,
creator,
tname, indextype, colnames, interval, level )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
WHERE user_id = SYSINDEX.creator ),
index_name,
( SELECT file_name FROM SYS.SYSFILE
WHERE file_id = SYSINDEX.file_id ),
( SELECT user_name FROM SYS.SYSUSERPERM
WHERE user_id = SYSINDEX.creator ),
table_name,
IF "unique" = 'Y' THEN 'Unique'
ELSE 'Non-unique' ENDIF,
( SELECT list( string( column_name,
IF "order" = 'A' THEN ' ASC' i
ELSE ' DESC' ENDIF ) )
FROM SYS.SYSIXCOL == SYS.SYSCOLUMN
WHERE index_id = SYSINDEX.index_id ), 0, 0
FROM SYS.SYSTABLE KEY JOIN SYS.SYSINDEX
```

Presents index information from SYSINDEX and SYSIXCOL in a readable format.

SYSOPTIONS system view

```
CREATE VIEW SYS.SYSOPTIONS ( user_name, "option",
"setting" )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
          WHERE user_id = SYSOPTION.user_id ),
          "option", "setting"
FROM SYS.SYSOPTION
```

Displays option settings contained in the table SYSOPTION in a readable format.

SYSPROCAUTH system view

```
CREATE VIEW SYS.SYSPROCAUTH(grantee,
       creator,procname)
AS
SELECT(select user_name FROM SYS.SYSUSERPERM
       WHERE SYSPROCPERM.grantee=SYSUSERPERM.user_id),
       (select user_name FROM SYS.SYSUSERPERM
       WHERE SYSPROCEDURE.creator=SYSUSERPERM.user_id),
       proc_name
FROM
SYS.SYSPROCEDURE JOIN SYS.SYSPROCPERM
```

Presents the procedure authorities from SYSUSERPERM in a readable format.

SYSPROCPARMS system view

```
CREATE VIEW SYS.SYSPROCPARMS ( creator, parmname,
procname,
parmtype, parmmode, parmdomain, length, remarks )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
           WHERE user_id = SYSPROCEDURE.creator ),
       parm_name, proc_name, parm_type,
       IF parm_mode_in = 'Y' AND
           parm_mode_out = 'N' THEN 'IN'
       ELSE IF parm_mode_in = 'N'
           AND parm_mode_out = 'Y' THEN 'OUT'
       ELSE 'INOUT' ENDIF ENDIF,
       ( SELECT domain_name FROM SYS.SYSDOMAIN
           WHERE domain_id = SYSPROCPARM.domain_id ),
       width, SYSPROCPARM.remarks
FROM SYS.SYSPROCPARM == SYS.SYSPROCEDURE
```

Lists all the procedure parameters from SYSPROCPARM in a readable format.

SYSPUBLICATIONS system view

```
CREATE VIEW SYS.SYSPUBLICATIONS
AS
SELECT(select user_name FROM SYS.SYSUSERPERM AS u
    WHERE u.user_id=p.creator) AS creator,
    publication_name,remarks
FROM SYS.SYSPUBLICATION AS p
```

Presents the user name from the SYSUSERPERM table for all creators, and displays the publication name and remarks from the SYSPUBLICATION table in a readable format.

SYSREMOTEOPTIONS system view

```
CREATE VIEW SYS.SYSREMOTEOPTIONS
AS
SELECT type_name,
    user_name,
    "option",
    setting
FROM SYS.SYSREMOTETYPE AS srt,
    SYS.SYSREMOTEOPTIONTYPE AS srot,
    SYS.SYSREMOTEOPTION AS sro,
    SYS.SYSUSERPERM AS sup
WHERE srt.type_id = srot.type_id
AND srot.option_id = sro.option_id
AND sro.user_id = sup.user_id
```

Presents the data from the system tables SYSREMOTEOPTION and SYSREMOTEOPTIONTYPE in a readable format.

SYSREMOTETYPES system view

```
CREATE VIEW SYS.SYSREMOTETYPES
AS SELECT type_id,type_name,publisher_address,remarks
FROM SYS.SYSREMOTETYPE
```

Presents the SQL Remote information from the SYSREMOTETYPE system table in a readable format.

SYSREMOTEUSERS system view

```
CREATE VIEW SYS.SYSREMOTEUSERS
AS SELECT(SELECT user_name FROM SYS.SYSUSERPERM AS u
      WHERE u.user_id=r.user_id) AS user_name,
    "consolidate",
    (SELECT type_name FROM SYS.SYSREMOTETYPE AS t
      WHERE t.type_id=r.type_id) AS type_name,
    "address",frequency,send_time,
    (IF frequency='A' THEN
      NULL
    ELSE
      IF frequency='P' THEN
        IF time_sent IS NULL THEN
          current timestamp
        ELSE
          (SELECT min(minutes(time_sent,
            60*hour(a.send_time)
            +minute(seconds(a.send_time,59))))
            FROM SYS.SYSREMOTEUSER AS a
            WHERE a.frequency='P'
            AND a.send_time=r.send_time)
        ENDIF
      ELSE
        IF current date+send_time
        >COALESCE(time_sent,current timestamp) THEN
          current date+send_time
        ELSE
          current date+send_time+1
        ENDIF
      ENDIF
    ENDIF) AS next_send,
    log_send,time_sent,log_sent,
    confirm_sent,send_count,resend_count,
    time_received,log_received,confirm_received,
    receive_count,rereceive_count
FROM SYS.SYSREMOTEUSER AS r
```

Lists the information in SYSREMOTEUSER in a readable format.

SYSSUBSCRIPTIONS system view

```
CREATE VIEW SYS.SYSSUBSCRIPTIONS
AS
SELECT(select publication_name
       FROM SYS.SYSPUBLICATION AS p
       WHERE p.publication_id=s.publication_id) AS
       publication_name,
       (select user_name FROM SYS.SYSUSERPERM AS u
       WHERE u.user_id=s.user_id) AS user_name,
       subscribe_by,created,started
FROM SYS.SYSSUBSCRIPTION AS s
```

Presents subscription information, such as the publication name, creation time, and start time from the SYSPUBLICATION table in a readable format.

SYSTABAUTH system view

```
CREATE VIEW SYS.SYSTABAUTH ( grantor, grantee,
    screator, stname, tcreator, ttname,
    selectauth, insertauth, deleteauth,
    updateauth, updatecols, alterauth, referenceauth )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
           WHERE user_id = SYSTABLEPERM.grantor ),
       ( SELECT user_name FROM SYS.SYSUSERPERM
           WHERE user_id = SYSTABLEPERM.grantee ),
       ( SELECT user_name
           FROM SYS.SYSUSERPERM == SYS.SYSTABLE
           WHERE table_id = SYSTABLEPERM.stable_id ),
       ( SELECT table_name FROM SYS.SYSTABLE
           WHERE table_id = SYSTABLEPERM.stable_id ),
       ( SELECT user_name FROM
           SYS.SYSUSERPERM == SYS.SYSTABLE
           WHERE table_id = SYSTABLEPERM.ttable_id ),
       ( SELECT table_name FROM SYS.SYSTABLE
           WHERE table_id = SYSTABLEPERM.ttable_id ),
       selectauth, insertauth, deleteauth,
       updateauth, updatecols,
       alterauth, referenceauthauth
FROM SYS.SYSTABLEPERM
```

Presents table permission information in SYSTABLEPERM in a readable format.

SYSUSERAUTH system view

```
CREATE VIEW SYS.SYSUSERAUTH ( name, password,
resourceauth, dbaauth,
scheduleauth, user_group )
AS
SELECT user_name, password, resourceauth,
       dbaauth, scheduleauth, user_group
FROM SYS.SYSUSERPERM
```

Displays all the information in the table SYSUSERPERM except for user numbers. Since this view shows passwords, this system view does not have PUBLIC select permission. (All other system views have PUBLIC select permission.)

SYSUSERLIST system view

```
CREATE VIEW SYS.SYSUSERLIST ( name, resourceauth,
   dbaauth, scheduleauth, user_group )
AS
SELECT user_name, resourceauth,
       dbaauth, scheduleauth, user_group
FROM SYS.SYSUSERPERM
```

Presents all information in SYSUSERAUTH except passwords.

SYSUSEROPTIONS system view

```
CREATE VIEW SYS.SYSUSEROPTIONS ( "user_name",
   "option", "setting" )
AS
SELECT u.name, "option",
       isnull( ( SELECT "setting"
```

```
                FROM sys.sysoptions s
        WHERE s.user_name = u.name
        AND s."option" = o."option" ),
        "setting" )
FROM SYS.SYSOPTIONS o, SYS.SYSUSERAUTH u
WHERE o.user_name = 'PUBLIC'
```

Displays effective permanent option settings for each user. If a user has no setting for an option, this view displays the public setting for the option.

SYSUSERPERMS system view

```
CREATE VIEW SYS.SYSUSERPERMS
AS
SELECT user_id, user_name, resourceauth, dbaauth,
       scheduleauth, user_group, remarks
FROM SYS.SYSUSERPERM
```

Contains exactly the same information as the table SYS.SYSUSERPERM except for the password. All users have read access to this view, but only the DBA has access to the underlying table (SYS.SYSUSERPERM).

SYSVIEWS system view

```
CREATE VIEW SYS.SYSVIEWS ( vcreator, viewname, viewtext
)
AS
SELECT user_name, table_name, view_def
FROM SYS.SYSTABLE KEY JOIN SYS.SYSUSERPERM
WHERE table_type = 'VIEW'
```

Lists views along with their definitions.

Transact-SQL compatibility view

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 11-1, Table 11-2, and Table 11-3 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 11-1: Tables in each ASE database

Table name	Description	Data?
sysalternates	One row for each user mapped to a database user	No
syscolumns	One row for each column in a table or view, and for each parameter in a procedure	Yes
syscomments	One or more rows for each view, rule, default, and procedure, giving SQL definition statement	Yes
sysconstraints	One row for each referential and check constraint associated with a table or column	No
sysdepends	One row for each procedure, view, or table that is referenced by a procedure, view	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	Yes
syskeys	One row for each primary, foreign, or common key; set by user (not maintained by Adaptive Server Enterprise)	No
syslogs	Transaction log	No
sysobjects	One row for each table, view, procedure, rule, default, log, and (in tempdb only) temporary object	Contains compatible data only
sysprocedures	One row for each view, rule, default, and procedure, giving internal definition	No
sysprotects	User permissions information	No
sysreferences	One row for each referential integrity constraint declared on a table or column	No
sysroles	Maps server-wide roles to local database groups	No
syssegments	One row for each segment (named collection of disk pieces)	No
systhresholds	One row for each threshold defined for the database	No
systypes	One row for each system-supplied and user-defined data type	Yes
sysusermessages	One row for each user-defined message	Yes (this is an IQ system table)
sysusers	One row for each user allowed in the database	Yes

Tables in the Adaptive Server Enterprise master database

Table 11-2: ASE master database tables

Table name	Description	Data?
syscharsets	One row for each character set or sort order	No
sysconfigures	One row for each configuration parameter that can be set by a user	No
syscurconfigs	Information about configuration parameters currently being used by the server	No
sysdatabases	One row for each database on the server	No
sysdevices	One row for each tape dump device, disk dump device, disk for databases, and disk partition for databases	No
sysengines	One row for each server currently online	No
syslanguages	One row for each language (except U.S. English) known to the server	No
syslocks	Information about active locks	No
sysloginroles	One row for each server login that possesses a system-defined role	No
syslogins	One row for each valid user account	Yes
sysmessages	One row for each system error or warning	No
sysprocesses	Information about server processes	No
sysremotelogins	One row for each remote user	No
syssrvroles	One row for each server-wide role	No
sysservers	One row for each remote server	No
sysusages	One row for each disk piece allocated to a database	No

Tables in the Adaptive Server Enterprise sybsecurity database

Table 11-3: ASE sybsecurity database tables

Table name	Description	Data?
sysaudits	One row for each audit record	No
sysauditoptions	One row for each global audit option	No

APPENDIX A

Compatibility with Other Sybase Databases

About this appendix

This appendix is provided 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 Adaptive Server Anywhere. Beginning with an overview of Transact-SQL, it compares these databases in several areas that you need to be aware of when 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 12.7 with Adaptive Server Enterprise 12.5.2 and Adaptive Server Anywhere 9.0.1.

Topics

Compatibility information elsewhere in this book

Compatibility information is also provided in the following chapters:

- In Chapter 2, “Database Options,”see “Transact-SQL compatibility options” on page 35.
- In Chapter 4, “SQL Data Types,” see compatibility information for each data type; also see Data type conversions on page 241.
- In Chapter 6, “SQL Statements,” see the compatibility information in each command.

A note on Adaptive Server Anywhere

Sybase IQ is an extension of Adaptive Server 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 Adaptive Server Anywhere documentation are supported for Sybase IQ.

The Sybase IQ documentation set calls out differences in many cases, but not all. Sybase IQ *documentation always supersedes the Adaptive Server Anywhere documentation.* Except for topics where the Sybase IQ documentation refers you to Adaptive Server Anywhere documentation, always refer to the documentation listed as “Documentation for Sybase IQ” in “About This Book,” immediately after the Table of Contents of each Sybase IQ book.

An overview of Transact-SQL support

Sybase IQ, like Adaptive Server 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 an Adaptive Server 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 together with the syntax supported in earlier versions of Sybase IQ. For these statements, Adaptive Server 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, Adaptive Server 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. Adaptive Server 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 Adaptive Server 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 Adaptive Server 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 no level corresponding 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, not to the server. When a user connects, they connect 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.

Adaptive Server 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.

For information about Open Server support, see the *Sybase IQ System Administration Guide*. Chapter 15, "Sybase IQ as a Data Server" in that book includes 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 allocate space initially 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.

- Adaptive Server 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. Adaptive Server 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 *Sybase IQ System Administration Guide*.

System tables, Catalog Store, and IQ Store

An IQ database is a joint data store consisting of three parts:

- The Catalog Store includes system tables and stored procedures, and resides in a set of tables that are compatible with Adaptive Server 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 these:

- Adaptive Server Anywhere and Sybase IQ use a different schema from Adaptive Server Enterprise for the catalog (tables, columns, and so on).
- Adaptive Server 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 9, “System Tables” and Chapter 11, “System Views.”
- In Adaptive Server Enterprise, the database owner (user ID dbo) owns the catalog objects.

- In Adaptive Server 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 Adaptive Server 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 Adaptive Server 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, you should 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 4, “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:

- Adaptive Server 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.

- Adaptive Server 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, but Sybase IQ does not.

When inserting into CHAR or VARCHAR:

- Adaptive Server 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.
- Adaptive Server Anywhere supports up to 32K-1 with CHAR and VARCHAR, and up to 2GB with LONG VARCHAR.
- Adaptive Server Anywhere supports the name LONG VARCHAR and its synonym TEXT, while Adaptive Server Enterprise only supports 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.

- Adaptive Server 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

Table A-1 summarizes binary data type support.

Table A-1: Binary data type supported sizes

Data type	Adaptive Server Enterprise	Adaptive Server 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 Adaptive Server Anywhere display binary data differently when projected:

- Sybase IQ supports both Adaptive Server Enterprise and Adaptive Server Anywhere display formats.
- If ‘123’ is entered in a BINARY field the Adaptive Server 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.

- Adaptive Server 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).
- Adaptive Server 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.
- With the option AUTOMATIC_TIMESTAMP set on, Adaptive Server Anywhere has the same behavior as Adaptive Server Enterprise in giving columns whose data type is timestamp an automatic default of timestamp if no other default is provided.

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.
- Adaptive Server 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.
- Adaptive Server 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.
- Adaptive Server 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, Adaptive Server Anywhere, and Sybase IQ have different default precision and scale:

- In Adaptive Server Enterprise, the default is precision 18 scale 0.
- In Adaptive Server 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, you should always specify a precision and scale for Sybase IQ exact numeric types.

Approximate numeric data types

Adaptive Server Enterprise differs from Adaptive Server Anywhere and Sybase IQ in how the FLOAT(p) data type is interpreted: that is, when to create a 4-byte data type, and when to create an 8-byte data type.

Adaptive Server Anywhere and Sybase IQ offer the FLOAT_AS_DOUBLE option to control data width.

Text data type

Support for TEXT data is currently implemented as follows:

- Adaptive Server Enterprise and Adaptive Server Anywhere support up to 2 GB 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 Adaptive Server 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. Adaptive Server 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 908 and “Space allocation and device management” on page 907 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's 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, passwords follow the case sensitivity of the data by default; however, you can also specify password case sensitivity independently of data case sensitivity. The default password for case-sensitive databases is uppercase SQL. The default user ID is DBA, but it is not 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 Adaptive Server Anywhere and Sybase IQ.

Table name uniqueness

Table name uniqueness requirements apply within a database:

- For Sybase IQ and Adaptive Server 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:

- Adaptive Server 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.
- Adaptive Server 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

If you have databases created with a previous version of Sybase IQ, run the stored procedure sp_iqprintconstraints to list all Sybase IQ tables and columns in a format that allows you to recreate them after deletion. If you want to drop all constraints on Sybase IQ tables in the database, you can then run the sp_iqdropconstraints procedure.

Adaptive Server Enterprise and Adaptive Server Anywhere enforce CHECK constraints. Adaptive Server 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 *Sybase IQ System Administration Guide*.

Actions for enforcing integrity are supported as follows:

- Adaptive Server 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 Adaptive Server Anywhere support specifying a default value for a column.
- Only Adaptive Server 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.
- Adaptive Server Anywhere supports AUTOINCREMENT default value. Adaptive Server 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. Adaptive Server 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. ASA identity columns can be altered to be nonidentity columns and vice versa. You can add or drop AUTOINCREMENT columns from ASA 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:

- Adaptive Server 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 on page 523.

For information about creating tables, see the CREATE TABLE statement on page 499.

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.
- Adaptive Server Anywhere and Sybase IQ support the CREATE DOMAIN statement to achieve the same objective.

CREATE TRIGGER statement

Support for triggers differs as follows:

- Adaptive Server 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 Adaptive Server 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 Adaptive Server Anywhere because ASA allows, but ignores, the keywords FILLFACTOR, IGNORE_DUP_KEY, SORTED_DATA, IGNORE_DUP_ROW, and ALLOW_DUP_ROW.

- Adaptive Server Anywhere CREATE INDEX syntax supports the VIRTUAL keyword for use by its Index Consultant, but not for actual execution of queries.

- 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 *Sybase IQ System Administration Guide* for more information on Sybase IQ indexes.

Users, groups, and permissions

There are some differences between the Adaptive Server Enterprise and Adaptive Server 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.

In Adaptive Server Anywhere and Sybase IQ, users connect directly to a database and do not require a server login ID. Instead, each user receives a user ID and password on a database so they can use that 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.
- Adaptive Server 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
- Adaptive Server 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.

- Adaptive Server 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.

Adaptive Server 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 accounts let DBAs control user connections and password expirations.

For details on Sybase IQ User Administration, see Chapter 12, “Managing User IDs and Permissions” and Chapter 15, “Sybase IQ as a Data Server” in the *Sybase IQ System Administration Guide*.

Although Adaptive Server 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 10, “System Procedures,” including “Adaptive Server Enterprise system procedures” on page 884.

Load formats

Load format support in the three products is as follows:

- Sybase IQ handles ASCII and BINARY load formats.
- Adaptive Server 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 Adaptive Server Anywhere LOAD statement is based on BCP and designed to offer exactly the same functionality.

BCP support in loading

- Adaptive Server Enterprise and Adaptive Server Anywhere support BCP in.
- Sybase IQ supports BCP through, iq_bcp, an o

  pen-client-based utility that copies a database table to or from an operating system file in a user-specified format.

You can perform a BCP into an Adaptive Server Anywhere table and then transfer the contents to Sybase IQ; however, the transfer of rows from Adaptive Server Anywhere to Sybase IQ is executed one row at a time. Sybase IQ does not support BLKLIB, so BCP, which uses Open Client's Bulk-Library, does not work in load mode. Both Sybase IQ and Adaptive Server Enterprise BCP format supports a blank when one digit is in the date.

Setting options for Transact-SQL compatibility

Set Sybase IQ database options using the SET OPTION statement. See "Transact-SQL compatibility options" on page 35 for a list of option settings required for Transact-SQL-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.
- Set the Sybase IQ database option PERCENT_AS_COMMENT OFF and use -- (double dash) as a comment delimiter for SQL92 compatibility. The percent character is the default comment delimiter in Sybase IQ. Adaptive Server Enterprise treats the % as a modulo operator and does not support the Sybase IQ mod function.

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 on page 632.

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 Adaptive Server Anywhere. Adaptive Server Enterprise and Adaptive Server Anywhere support subqueries in the ON clause; Sybase IQ does not currently support this.

UNION in subqueries is supported as follows:

- Adaptive Server 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.

Adaptive Server 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.
- Adaptive Server Anywhere does not support the GROUP BY ALL Transact-SQL extension.

- Sybase IQ supports the GROUP BY ALL Transact-SQL extension on a single table only, not in a view.

ROLLUP and CUBE in the GROUP BY clause are supported as follows:

- Sybase IQ and Adaptive Server 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 extended group by semantics with some exceptions. For details, see "GROUP BY" on page 637. Adaptive Server Anywhere does not support extended group by semantics. Only Adaptive Server Anywhere supports the List() aggregate to return a list of values.

For information about using GROUP BY with OLAP functions, see Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*

COMPUTE clause

COMPUTE clause support is as follows:

- Adaptive Server Enterprise supports the Transact-SQL COMPUTE clause.
- Adaptive Server 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.

- Adaptive Server 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.
- Adaptive Server 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 T-SQL outer join syntax is deprecated in Adaptive Server 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 Adaptive Server Anywhere support subqueries in the ON clause.

Full outer join support is as follows:

- Adaptive Server 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, refer to the white paper "Semantics and Compatibility of Transact-SQL Outer Joins," which is available on the Sybase online support Web site MySybase at http://www.sybase.com/support/. Although written for Adaptive Server 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`.

Adaptive Server 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 Adaptive Server Anywhere 8.0 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.

- Adaptive Server 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, FOR BROWSE syntax.
- Adaptive Server 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 Adaptive Server Anywhere and Adaptive Server Enterprise, with these differences:

- Adaptive Server Enterprise supports the USING CHARACTERS | USING BYTES syntax in PatIndex(); Adaptive Server Anywhere and Sybase IQ do not.
- Adaptive Server Enterprise supports the Reverse() function; Adaptive Server Anywhere and Sybase IQ do not.
- Adaptive Server Enterprise supports Len() as alternative syntax for Length(); Adaptive Server Anywhere does not support this alternative.
- Adaptive Server Enterprise supports the Square() and Str_Replace() Microsoft compatibility functions; Adaptive Server Anywhere does not.
- Sybase IQ supports Str_Replace().
- Adaptive Server Enterprise and Adaptive Server 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 Adaptive Server Anywhere do not.

- Adaptive Server 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.

- Adaptive Server Anywhere and Sybase IQ support Lcase() and Ucase() as synonyms of Lower() and Upper(); Adaptive Server Enterprise does not.
- Adaptive Server Anywhere and Sybase IQ support the Locate() string function; Adaptive Server Enterprise does not.
- Adaptive Server 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.
- Adaptive Server Anywhere supports the NEWID, STRTOUID, and UUIDTOSTR functions; Adaptive Server Enterprise does not. You use these functions in Sybase IQ, but CIS functional compensation performance considerations apply.

Note Some SQL functions, including SOUNDEX and DIFFERENCE string functions, and some date functions operate differently in Sybase IQ and Adaptive Server 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: Dense_Rank(), Grouping(), Ntile(), Percent_Rank(), Percentile_Cont(), Percentile_Disc(), Rank(), StdDev(), Stddev_Pop, Stddev_Samp, Var_Pop, Var_Samp, and Variance().

Adaptive Server Anywhere supports all of the Sybase IQ OLAP functions, plus Corr(), Covar_Pop(), Covar_Samp(), Cume_Dist , Regr_Avgx(), Regr_Avgy(), Regr_Intercept(), Regr_Slope(), Regr_Sxx(), Regr_Sxy(), and Regr_Syy().

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. For more information, see Chapter 5, "SQL Functions." Also see Chapter 4, "Using OLAP" in the *Sybase IQ Performance and Tuning Guide*.

System functions

Adaptive Server Anywhere and Sybase IQ do not support the following Adaptive Server Enterprise system functions as they are specific to Adaptive Server Enterprise administration:

- 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 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:

- Adaptive Server Anywhere support 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

Adaptive Server 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.
- Adaptive Server 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 Adaptive Server Anywhere are more liberal than ANSI permits on which view definitions are updatable when the WITH CHECK option is not requested.

Adaptive Server 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.

Adaptive Server 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

Based on the ISO/ANSI draft standard, the Adaptive Server 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. Adaptive Server 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 Adaptive Server 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 Adaptive Server 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. Adaptive Server 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, Adaptive Server 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] on page 610.

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 Chapter 6, “SQL Statements” for details of individual statements.

Expression subqueries in IF statements

Adaptive Server Enterprise and Adaptive Server 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 statementS

Permitted usage of the CASE statement differs in Sybase IQ and Adaptive Server 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 179.

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 Adaptive Server 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, Adaptive Server 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 employee')
```

Using Sybase Central to translate stored procedures

Sybase Central has facilities for creating, viewing, and altering procedures.

❖ **Translating a stored procedure using Sybase Central**

1 Connect to a database using Sybase Central, either as owner of the procedure you want to change, or as a DBA user.

2 Double-click the Procedures folder to see a list of the stored procedures in the database.

3 Right-click the procedure you want to translate, and choose the target dialect from the submenu: either Watcom-SQL or Transact-SQL.

The procedure appears in the selected dialect. If the selected dialect is not the one in which the procedure is stored, the server translates it to that dialect. Any untranslated lines appear as comments.

4 Rewrite any untranslated lines as needed, and click Execute Script to save the translated version to the database. You can also export the text to a file for editing outside Sybase Central.

Returning result sets from Transact-SQL procedures

Adaptive Server 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 employee.emp_lname, employee.emp_fname
    FROM department, employee
    WHERE department.dept_name = @deptname
    AND department.dept_id = employee.dept_id
```

Example of Watcom-SQL procedure

The following is the corresponding Adaptive Server Anywhere or Sybase IQ procedure:

```
CREATE PROCEDURE showdept(in deptname varchar(30))
RESULT ( lastname char(20), firstname char(20))
BEGIN
    SELECT employee.emp_lname, employee.emp_fname
    FROM department, employee
    WHERE department.dept_name = deptname
    AND department.dept_id = employee.dept_id
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 8, "Using Procedures and Batches" in the *Sybase IQ System Administration Guide.*

Variables in Transact-SQL procedures

Adaptive Server 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 Adaptive Server 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 927. For more information on using the SET statement to assign variables, see SET statement on page 641.

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] on page 616.

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.

Adaptive Server Anywhere and Sybase IQ

The preceding sections, while focused on compatibility with Transact-SQL, also clarify many of the distinctions between Sybase IQ and Adaptive Server Anywhere.

This section points out other differences between Sybase IQ and Adaptive Server Anywhere.

For additional information, always refer to the Sybase IQ documentation set when using the product. Refer to the Adaptive Server Anywhere documentation set when using Adaptive Server Anywhere, or when the Sybase IQ documentation refers to Adaptive Server AnywhereAdaptive Server AnywhereAdaptive Server Anywhere documentation for specific functionality only.

Server and database start-up and administration

Note the following differences in starting and managing databases and servers:

- Sybase IQ uses the server start-up command start_asiq, instead of the Adaptive Server Anywhere network server start-up command.
- Sybase IQ does not support personal servers.
- Sybase IQ supports many Adaptive Server Anywhere server command line options but not all. Other server options are supported for Sybase IQ but not for Adaptive Server Anywhere.
- Sybase IQ provides the stop_asiq utility to shut down servers.
- Clauses permitted in the BACKUP and RESTORE statements differ in Sybase IQ and Adaptive Server Anywhere.
- SQL Remote is supported in Sybase IQ for multiplex operations only.

Sybase IQ supports many Adaptive Server Anywhere database administration utilities, but not all.

- The following Adaptive Server 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 Adaptive Server Anywhere validation utility on the Catalog Store only. To validate the IQ Store, use sp_iqcheckdb.

Database options

Some Adaptive Server Anywhere database options are not supported by Sybase IQ, including: DEFAULT_TIMESTAMP_INCREMENT and JAVA_INPUT_OUTPUT.

Some database options apply only to the Catalog Store, including: FOR_XML_NULL_TREATMENT, ISOLATION_LEVEL, PREFETCH, PRECISION, SCALE, and TRUNCATE_WITH_AUTO_COMMIT.

Options with differences in behavior, default, or allowed values include DELAYED_COMMITS, JAVA_HEAP_SIZE, TIME_FORMAT, TIMESTAMP_FORMAT.

Sybase IQ also includes many options that Adaptive Server Anywhere does not support. For details, see Chapter 2, “Database Options.”

Data definition language (DDL)

In addition to the DDL differences discussed previously, note these:

- 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). Adaptive Server 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 Adaptive Server Anywhere supports with the CHECK ON COMMIT clause of CREATE TABLE statement.
- A Sybase IQ table cannot have a foreign key that references an Adaptive Server Anywhere (or Catalog) table, and an Adaptive Server Anywhere table cannot have a foreign key that references a Sybase IQ table.
- 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 Adaptive Server 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 *Sybase IQ System Administration Guide*.

- Sybase IQ supports the INSERT...LOCATION syntax; Adaptive Server Anywhere does not.
- LOAD TABLE options differ in Sybase IQ and Adaptive Server 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 Adaptive Server Anywhere. For details, see Chapter 10, "Transactions and Versioning" in the *Sybase IQ System Administration Guide*.
- When you SELECT from a stored procedure in Sybase IQ, CIS functional compensation performance considerations apply. For more information, see "Conditions that cause processing by Adaptive Server Anywhere" in *Sybase IQ Performance and Tuning Guide.*

Index

A

B

C

D

E

F

G

H

I

J

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z