SYBASE®

Reference: Statements and Options

Sybase® IQ

15.1

DOCUMENT ID: DC00801-01-1510-01

LAST REVISED: July 2009

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

Contents

About This Book

Subject

This book provides reference material for Sybase IQ SQL statements and database options. Reference material for other aspects of Sybase IQ, including language elements, data types, functions, system procedures, and system tables is provided in *Reference: Building Blocks, Tables, and Procedures*. 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 *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 statements and options, but it does not describe why you might want to use these features. Use this book 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 *Sybase IQ Release Bulletin for Windows*.

Related documents

The Sybase IQ 15.1 documentation set includes:

- *Release Bulletin* provides information about last-minute changes to the product and documentation.
- *Installation and Configuration Guide* provides platform-specific instructions on installing, migrating to a new version, and configuring Sybase IQ for a particular platform.
- *Advanced Security in Sybase IQ* covers the use of user encrypted columns within the Sybase IQ data repository. You need a separate license to install this product option.
- *Error Messages* lists Sybase IQ error messages referenced by Sybase error code, SQLCode, and SQLState, and SQL preprocessor errors and warnings.

- *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library* contains a concise description of the IMSL C Stat Library time series C functions. This book is only available to RAP – The Trading Edition™ Enterprise users.
- *Introduction to Sybase IQ* includes hands-on exercises for those unfamiliar with Sybase IQ or with the Sybase Central™ database management tool.
- *Large Objects Management in Sybase IQ* explains storage and retrieval of Binary Large Objects (BLOBs) and Character Large Objects (CLOBs) within the Sybase IQ data repository. You need a separate license to install this product option.
- *New Features in Sybase IQ 15.0* documents new features and behavior changes for version 15.0.
- *New Features Summary Sybase IQ 15.1* summarizes new features and behavior changes for the current version.
- *Performance and Tuning Guide* describes query optimization, design, and tuning issues for very large databases.
- *Quick Start* lists steps to build and query the demo database provided with Sybase IQ for validating the Sybase IQ software installation. Includes information on converting the demo database to multiplex.
- *Reference Manual* – Includes two reference guides to Sybase IQ:
    - *Reference: Building Blocks, Tables, and Procedures* describes SQL, stored procedures, data types, and system tables that Sybase IQ supports.
    - *Reference: Statements and Options* describes the SQL statements and options that Sybase IQ supports.
- *System Administration Guide* – Includes two volumes:
    - *System Administration Guide: Volume 1* describes startup, connections, database creation, population and indexing, versioning, collations, system backup and recovery, troubleshooting, and database repair.
    - *System Administration Guide: Volume 2* describes writing and running procedures and batches, programming with OLAP, accessing remote data, setting up IQ as an Open Server, scheduling and event handling, programming with XML, and debugging.

- *User-Defined Functions Guide* provides information about the user-defined functions, their parameters, and possible usage scenarios.
- *Using Sybase IQ Multiplex* tells how to use multiplex capability, designed to manage large query loads across multiple nodes.
- *Utility Guide* provides Sybase IQ utility program reference material, such as available syntax, parameters, and options.

Sybase IQ and SQL Anywhere

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

Documentation for SQL Anywhere includes:

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

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

Documentation for Sybase Software Asset Management (SySAM) includes:

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

Other sources of information

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

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

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

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

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ Finding the latest information on product certifications

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

❖ Finding the latest information on component certifications

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

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

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

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ Finding the latest information on EBFs and software maintenance

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.
4. Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

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

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

Syntax conventions

This documentation uses these conventions in syntax descriptions:

- **Keywords** SQL keywords are shown in UPPERCASE. However, SQL keywords are case-insensitive, so you can enter keywords in any case; SELECT, Select, and select are equivalent.
- **Placeholders** Items that must be replaced with appropriate identifiers or expressions are shown in *italics*.
- **Continuation** Lines beginning with an ellipsis (...) are a continuation of the statements from the previous line.
- **Repeating items** Lists of repeating items are shown with an element of the list followed by an ellipsis (...). One or more list elements are allowed. If multiple elements are specified, they must be separated by commas.
- **Optional portions** Optional portions of a statement are enclosed by square brackets. For example:

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

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

  ```
  [ ASC | DESC ]
  ```

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

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

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

Typographic conventions

Table 1 lists the typographic conventions used in this documentation.

Table 1: Typographic conventions

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

The demo database

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

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

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

Accessibility features

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

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

Configuring your accessibility tool

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

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

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

If you need help

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

CHAPTER 1 **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 the sections “Identifiers,” “Search conditions,” “Expressions,” and “Strings” in Chapter 2, “SQL Language Elements” in *Reference: Building Blocks, Tables, and Procedures*.

- 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” in Chapter 2, “SQL Language Elements” in *Reference: Building Blocks, Tables, and Procedures*.
- 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 200.
- 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 2, “SQL Language Elements” in *Reference: Building Blocks, Tables, and Procedures*.

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:\\IQ-15_1\\sample.db");

   EXEC SQL ALLOCATE DESCRIPTOR sqlda1 WITH MAX 25;

   EXEC SQL PREPARE :stmt FROM
        'select * from Employees';
   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 156

"The SQL descriptor area (SQLDA)" in *SQL Anywhere Server – Programming > SQL Anywhere Data Access APIs > SQL Anywhere embedded SQL*

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 15.0 to 15.1.

Note See the *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 *Installation and Configuration Guide* for your platform.

After using ALTER DATABASE UPGRADE, shut down the database.

Note Use the iqunload utility to upgrade databases created prior to version 15.0. For details, see Chapter 6, “Migrating Data” in the *Installation and Configuration Guide* for your platform.

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 68

"Migrating Data" in the *Installation and Configuration Guide* for your platform

"Introduction to Java support" in *SQL Anywhere Server – Programming > Java in the database > Java support in SQL Anywhere*

ALTER DBSPACE statement

Description

Changes the read/write mode, changes the size, or extends an existing dbspace.

Syntax

ALTER DBSPACE *dbspace-name*
{ **ADD** *new-file-spec* [, new-file-spec ...]
| **DROP FILE** *logical-file-name* [, **FILE** *logical-file-name* ...]
| **RENAME TO** *newname* | **RENAME** '*new-file-pathname*'
| **READONLY** | **READWRITE**
| **ONLINE** | **OFFLINE**
| **STRIPING**{ **ON** | **OFF** }
| **STRIPESIZEKB** *size-in-KB*
ALTER FILE *file-name*
{ **READONLY** | **READWRITE** }
| **SIZE** *file-size* [KB | MB | GB | TB | PAGES]
| **ADD** *file-size* [KB | MB | GB | TB | PAGES] }
SERVER "*server-name*"] **RENAME PATH** '*new-file-pathname*'
RENAME TO *newname*

Parameters

new-file-spec:

FILE *logical-file-name 'file-path' iq-file-opts*

iq-file-opts:

[[**SIZE**] *file-size*]
…[**KB** | **MB** | **GB** | **TB**]]
[**RESERVE** *reserve-size* [KB | MB | GB | TB]]

Examples

Example 1 Change the mode of a dbspace called *DspHist* to READONLY.

```
ALTER DBSPACE DspHist READONLY
```

Example 2 Add 500MB to the dbspace FileHist3.

```
ALTER DBSPACE DspHist
ALTER FILE FileHist3 ADD 500MB
```

Example 3 On Solaris, add two 500MB files to the dbspace DspHist.

```
ALTER DBSPACE DspHist ADD
FILE FileHist3 '/History1/data/file3' SIZE 500MB
FILE FileHist3 '/History1/data/file4' SIZE 500
```

Example 4 Increase the size of the dbspace IQ_SYSTEM_TEMP by 2GB.

```
ALTER DBSPACE IQ_SYSTEM_TEMP ADD 2 GB
```

Example 5 Remove two files from dbspace DspHist. Both files must be empty.

```
ALTER DBSPACE DspHist
DROP FILE FileHist2, FILE FileHist4
```

Example 6 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 online/offline state, alters the file size, renames the dbspace name, file logical name or file path, or sets the dbspace striping parameters. For details about existing dbspaces, run sp_iqdbspace procedure, sp_iqdbspaceinfo procedure, sp_iqfile procedure, sp_iqdbspaceobjectinfo, and sp_iqobjectinfo. Dbspace and dbfile names are always case insensitive. The physical file paths are case sensitive, if the database is CASE RESPECT and the operating system supports case sensitive files. Otherwise, the file paths are case insensitive.

ADD FILE clause Adds one or more files to the specified dbspace. The dbfile name and the physical file path are required for each file and must be unique. You can add files to dbspaces of IQ main or IQ temporary dbspaces. You may add a file to a read-only dbspace, but the dbspace remains read-only.

A catalog dbspace may contain only one file, so ADD FILE may not be used on catalog dbspaces.

DROP FILE clause Removes the specified file(s) from an IQ dbspace. The file must be empty. You cannot drop the last file from the specified dbspace. Instead use DROP DBSPACE if the dbspace contains only one file.

RENAME TO clause
Renames the *dbspace-name* to a new name. The new name must be unique in the database. You cannot rename IQ_SYSTEM_MAIN, IQ_SYSTEM_MSG, IQ_SYSTEM_TEMP or SYSTEM.

RENAME clause Renames the pathname of the dbspace that contains a single file. It is semantically equivalent to the ALTER FILE RENAME PATH clause. An error is returned if the dbspace contains more than one file.

READONLY clause   Changes any dbspace except IQ_SYSTEM_MAIN, IQ_SYSTEM_TEMP, IQ_SYSTEM_MSG, and SYSTEM to read-only. Disallows DML modifications to any object currently assigned to the dbspace. Can only be used for dbspaces in the IQ main store.

READWRITE clause   Changes the dbspace to read-write. The dbspace must be online. Can only be used for dbspaces in the IQ main store.

ONLINE clause   Puts an offline dbspace and all associated files online. Can only be used for dbspaces in the IQ main store.

OFFLINE clause   Puts an online read-only dbspace and all associated files offline. (Returns an error if the dbspace is read-write, offline already, or not of the IQ main store.) Can only be used for dbspaces in the IQ main store.

STRIPING clause   Changes the disk striping on the dbspace as specified. When disk striping is set ON, data is allocated from each file within the dbspace in a round-robin fashion. For example, the first database page written goes to the first file, the second page written goes to the next file within given dbspace, and so on. Read-only dbspaces are skipped.

STRIPESIZEKB clause   Specifies the number of kilobytes (KB) to write to each file before the disk striping algorithm moves to the next stripe for the specified dbspace.

ALTER FILE READONLY   Changes the specified file to read-only. The file must be associated with an IQ main dbspace.

ALTER FILE READWRITE   Changes specified IQ main or temporary store dbfile to read-write. The file must be associated with an IQ main or temporary dbspace.

ALTER FILE SIZE clause   Specifies the new size of the file in units of 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 and if the dbspace has sufficient reserved space. You can decrease the size of the dbspace only if the portion to be truncated is not in use.

ALTER FILE ADD clause   Extends the size of the file in units of pages, kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). The default is MB. You can ADD only if the free list (an allocation map) has sufficient room and if the dbspace has sufficient reserved space.

You can also view and change the dbspace mode and size through the Sybase Central Dbspaces window.

ALTER FILE RENAME PATH clause Renames the file pathname associated with the specified file. This clause merely associates the file with the new file path instead of the old path. The clause does not actually change the operating system file name. You must change the file name through your operating system. The dbspace must be offline to rename the file path. The new path is used when the dbspace is altered online or when the database is restarted.

You may not rename the path of a file in IQ_SYSTEM_MAIN, because if the new path were not accessible, the database would be unable to start. If you need to rename the path of a file in IQ_SYSTEM_MAIN, make the file read-only, empty the file, drop the file, and add the file again with the new file path name.

ALTER FILE RENAME TO clause Renames the specified file's logical name to a new name. The new name must be unique in the database.

Side effects

- Automatic commit
- Automatic checkpoint
- A mode change to READONLY 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 81

CREATE DATABASE statement on page 68

DROP statement on page 177

sp_iqdbspace procedure in Chapter 7, "System Procedures" in *Reference: Building Blocks, Tables, and Procedures*

"Working with dbspaces" in Chapter 5, "Working with Database Objects," of the *System Administration Guide: Volume 1*

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 84

Chapter 3, “SQL Data Types” in *Reference: Building Blocks, Tables, and Procedures*

“SYSUSERTYPE system view”in Chapter 8, “System Views” in *Reference: Building Blocks, Tables, and Procedures*

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 CREATE EVENT statement on page 86.

Side effects

Automatic commit.

Permissions

Must have DBA authority.

See also

BEGIN … END statement on page 47

CREATE EVENT statement on page 86

Chapter 6, “Automating Tasks Using Schedules and Events” in the *System Administration Guide: Volume 2*

ALTER FUNCTION statement

Description

Modifies an existing function. You must include the entire new function in the ALTER FUNCTION statement.

Syntax

Syntax 1

ALTER FUNCTION [*owner*.]*function-name function-definition*

function-definition : **CREATE FUNCTION** syntax

Syntax 2

ALTER FUNCTION [*owner*.]*function-name*
SET HIDDEN

Syntax 3

ALTER FUNCTION [*owner*.]*function-name*
RECOMPILE

Usage

Syntax 1 Identical in syntax to the CREATE FUNCTION statement except for the first word. Either version of the CREATE FUNCTION statement can be altered.

Existing permissions on the function are maintained and do not have to be reassigned. If a DROP FUNCTION and CREATE FUNCTION were carried out, execute permissions must be reassigned.

Syntax 2 Use SET HIDDEN to scramble the definition of the associated function and cause it to become unreadable. The function can be unloaded and reloaded into other databases.

Warning! The SET HIDDEN setting is irreversible. If you need the original source again, you must maintain it outside the database.

If you use SET HIDDEN, debugging using the stored procedure debugger does not show the function definition, nor is it be available through procedure profiling.

Syntax 3 Use RECOMPILE to recompile a user-defined function. When you recompile a function, the definition stored in the catalog is re-parsed and the syntax is verified. The preserved source for a function is not changed by recompiling. When you recompile a function, the definitions scrambled by the SET HIDDEN clause remain scrambled and unreadable.

Side Effects

Automatic commit.

Standards

- **SQL2003** Vendor extension.

Permissions

Must be the owner of the function or have DBA authority.

See also

ALTER PROCEDURE statement on page 20

CREATE FUNCTION statement on page 97

DROP statement on page 177

"Hiding the contents of procedures, functions, and views" in Chapter 1, "Using Procedures and Batches" in the *System Administration Guide: Volume 2*

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

ALTER { **INDEX** *index-name*
| **[INDEX] FOREIGN KEY** *role-name*
| **[INDEX] PRIMARY KEY**
| **ON** [*owner*.]*table-name* { *rename-clause* | *move-clause* }

Parameters

rename-clause
:
RENAME TO | AS *new-name*

move-clause:
MOVE TO *dbspace-name*

Examples

Example 1 The following statement moves the primary key, HG for c5, from dbspace Dsp4 to Dsp8.

```
CREATE TABLE foo
    c1 INT IN Dsp1,
    c2 VARCHAR(20),
    c3 CLOB IN Dsp2,
    c4 DATE,
    c5 BIGINT,
    PRIMARY KEY (c5) IN Dsp4) IN Dsp3;

CREATE DATE INDEX c4_date ON foo(c4) IN Dsp5;

ALTER INDEX PRIMARY KEY ON foo MOVE TO Dsp8;
```

Example 2 Moves DATE index from Dsp5 to Dsp9

```
ALTER INDEX c4_date ON foo MOVE TO Dsp9
```

Example 3 Renames an index COL1_HG_OLD in the table jal.mytable to COL1_HG_NEW:

```
ALTER INDEX COL1_HG_OLD ON jal.mytable
RENAME AS COL1_HG_NEW
```

Example 4 Renames a foreign key role name ky_dept_id in table dba.Employees to emp_dept_id:

```
ALTER INDEX FOREIGN KEY ky_dept_id
ON dba.Employees
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 clause The ON clause specifies the name of the table that contains the index or foreign key to rename.

RENAME [AS | TO] clause The RENAME clause specifies the new name of the index or foreign key role.

MOVE clause The MOVE clause moves the specified index, unique constraint, foreign key, or primary key to the specified dbspace. For unique constraint or foreign key, you must specify its unique index name.

You must have DBA authority or have CREATE privilege on the new dbspace and be the table owner

---

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 25

CREATE INDEX statement on page 105

CREATE TABLE statement on page 135

ALTER LOGIN POLICY statement

Description

Modifies some or all option values for existing login policies in the database.

Syntax

ALTER LOGIN POLICY *policy-name policy-options*
[**MULTIPLEX SERVER** *server-name*]

Parameters

policy-options

policy-option [*policy-option*...]

policy_option:

policy-option-name =*policy-option-value*

policy-option-value={ **UNLIMITED** | **ROOT** | *value* }

Examples

The following example alters the Test1 login policy. This example changes the locked and max_connections options. The locked value indicates that users with the policy are prohibited from establishing new connections and the max_connections value indicates the number of concurrent connections allowed.

```
ALTER LOGIN POLICY Test1
locked=ON
max_connections=5;
```

Usage

For descriptions of login policy options, see CREATE LOGIN POLICY statement on page 117.

When a login policy is altered, changes are immediately applied to all users.

Permissions

Must have DBA authority.

See also

"Login management" in Chapter 8, "Managing User IDs and Permissions," in the *System Administration Guide: Volume 1*

ALTER PROCEDURE statement

Description

Replaces an existing 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 120

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** }]
[**CONNECTION CLOSE** [**CURRENT** | **ALL** | *connection-id*]]

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

Example 1 Changes the server class of the Adaptive Server Enterprise 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'
```

Example 2 Changes a capability of server infodc:

```
ALTER SERVER infodc
CAPABILITY 'insert select' OFF
```

Example 3 The following example closes all connections to the remote server named rem_test.

```
ALTER SERVER rem_test
CONNECTION CLOSE ALL
```

Example 4 The following example closes the connection to the remote server named rem_test that has the connection ID 142536.

```
ALTER SERVER rem_test
CONNECTION CLOSE 142536
```

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 4, “Accessing Remote Data” and Chapter 5, “Server Classes for Remote Data Access” in the *System Administration Guide: Volume 2*.

USING clause The USING clause changes the server’s connection information. For more information about connection information, see CREATE SERVER statement on page 130.

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.

CONNECTION CLOSE clause When a user creates a connection to a remote server, the remote connection is not closed until the user disconnects from the local database. The CONNECTION CLOSE clause allows you to explicitly close connections to a remote server. You may find this useful when a remote connection becomes inactive or is no longer needed.

The following SQL statements are equivalent and close the current connection to the remote server:

```
ALTER SERVER server-name CONNECTION CLOSE
```

```
ALTER SERVER server-name CONNECTION CLOSE CURRENT
```

You can close both ODBC and JDBC connections to a remote server using this syntax. You do not need DBA authority to execute either of these statements.

You can also disconnect a specific remote ODBC connection by specifying a connection ID, or disconnect all remote ODBC connections by specifying the ALL keyword. If you attempt to close a JDBC connection by specifying the connection ID or the ALL keyword, an error occurs. When the connection identified by *connection-id* is not the current local connection, the user must have DBA authority to be able to close the connection.

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 DBA authority to execute this command.

See also

CREATE SERVER statement on page 130

DROP SERVER statement on page 183

Chapter 4, "Accessing Remote Data" and Chapter 5, "Server Classes for Remote Data Access" in the *System Administration Guide: Volume 2*

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.ISYSTAB
```

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 132

DROP SERVICE statement on page 184

"Using the Built-in Web Server" in *SQL Anywhere Server – Database Administration*

ALTER TABLE statement

Description

Modifies a table definition.

Syntax

ALTER TABLE [*owner*.]*table-name*
{ *alter-clause*, ... }

Parameters

alter-clause:
ADD *create-clause*
| ALTER *column-name* *column-alteration*
| ALTER [CONSTRAINT *constraint-name*] CHECK (*condition*)
| DROP *drop-object*
| RENAME *rename-object*
| *move-clause*
| SPLIT PARTITION *partition-name* INTO (*partition-decl-1*, *partition-decl-2*)
| MERGE PARTITION *partition-name-1* INTO *partition-name-2*
| UNPARTITION
| PARTITION BY RANGE (*partition-key*) *range-partition-decl*

create-clause:
column-name column-definition [*column-constraint*]
| *table-constraint*
| PARTITION BY *partitioning-schema*

column-alteration:
{ *column-data-type* | *alterable-column-attribute* } [*alterable-column-attribute…*]
| ADD [*constraint-name*] CHECK (*condition*)
| DROP { DEFAULT | CHECK | CONSTRAINT *constraint-name* }

alterable-column-attribute:
[NOT] NULL
| DEFAULT *default-value*
| [CONSTRAINT *constraint-name*] CHECK { NULL | (*condition*) }

column-constraint:
[CONSTRAINT *constraint-name*] { UNIQUE
| PRIMARY KEY
| REFERENCES *table-name* [(*column-name*)] [*actions*]
| CHECK (*condition*)
| IQ UNIQUE (*integer*) }

drop-object:
{ *column-name*
| CHECK|
| CONSTRAINT *constraint-name*
| UNIQUE (*index-columns-list*)
| PRIMARY KEY
| FOREIGN KEY *fkey-name*
| PARTITION *range-partition-name*
}

move-clause:
{ ALTER *column-name* MOVE
{ PARTITION (*partition-name* TO *new-dbspace-name*)
| TO *new-dbspace-name* } }
| MOVE PARTITION *partition-name* TO *new-dbspace-name*
| MOVE TO *new-dbspace-name*
| MOVE METADATA TO *new-dbspace-name*

rename-object:
new-table-name
| *column-name* TO *new-column-name*
| CONSTRAINT *constraint-name* TO *new-constraint-name*
| PARTITION *partition-name* TO *new-partition-name*

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
| NULL
| TIMESTAMP
| LAST USER
| USER

special-value:
CURRENT { DATABASE | DATE | REMOTE USER | TIME
| TIMESTAMP | USER | PUBLISHER }

table-constraint:
[CONSTRAINT *constraint-name*]
{ 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*] [

rename-object:
new-table-name | *column-name* TO *new-column-name*
| CONSTRAINT *constraint-name* TO *new-constraint-name* | PARTITION *partition-name* TO *new-partition-name*

range-partitioning-scheme:
RANGE(*partition-key*)
(
range-partition-decl [,*range-partition-decl* ...]
)

partition-key:
column-name

range-partition-decl:
partition-name VALUES <= ({*constant* | MAX }) [IN *dbspace-name*]

actions:
[ON { UPDATE | DELETE } *action*]

action:
{ RESTRICT }

Examples

Example 1 Adds a new column to the *Employees* table showing which office they work in:

```
ALTER TABLE Employees
ADD office CHAR(20)
```

Example 2 Drops the *office* column from the *Employees* table:

```
ALTER TABLE Employees
DROP office
```

Example 3 Adds a column to the *Customers* table assigning each customer a sales contact:

```
ALTER TABLE Customers
ADD SalesContact INTEGER
REFERENCES Employees (EmployeeID)
```

Example 4 Adds a new column *CustomerNum* to the *Customers* table and assigns a default value of 88:

```
ALTER TABLE Customers
ADD CustomerNum INTEGER DEFAULT 88
```

Example 5 Only FP indexes for c2, c4 and c5, are moved from dbspace Dsp3 to Dsp6. FP index for c1 remains in Dsp1. FP index for c3 remains in Dsp2. The primary key for c5 remains in Dsp4. Date index c4_date remains in Dsp5.

```
CREATE TABLE foo (
       c1 INT IN Dsp1,
       c2 VARCHAR(20),
       c3 CLOB IN Dsp2,
       c4 DATE,
       c5 BIGINT,
       PRIMARY KEY (c5) IN Dsp4) IN Dsp3;

    CREATE DATE INDEX c4_date ON foo(c4) IN Dsp5;
    ALTER TABLE foo
       MOVE TO Dsp6;
```

Example 6 Moves only FP index c1 from dbspace Dsp1 to Dsp7.

```
ALTER TABLE foo ALTER c1 MOVE TO Dsp7
```

Example 7 This example illustrates the use of many ALTER TABLE clauses to move, split, rename, and merge partitions.

Create a partitioned table:

```
CREATE TABLE bar (
       c1 INT,
       c2 DATE,
       c3 VARCHAR(10))
    PARTITION BY RANGE(c2)
    (p1 VALUES <= ('2005-12-31') IN dbsp1,
     p2 VALUES <= ('2006-12-31') IN dbsp2,
     P3 VALUES <= ('2007-12-31') IN dbsp3,
     P4 VALUES <= ('2008-12-31') IN dbsp4);

    INSERT INTO bar VALUES(3, '2007-01-01', 'banana
nut');
    INSERT INTO BAR VALUES(4, '2007-09-09', 'grape
jam');
    INSERT INTO BAR VALUES(5, '2008-05-05', 'apple
cake');
```

Move partition p2 to dbsp5:

```
ALTER TABLE bar MOVE PARTITION p2 TO DBSP5;
```

Split partition p4 into 2 partitions:

```
ALTER TABLE bar SPLIT PARTITION p4 INTO
       (P41 VALUES <= ('2008-06-30') IN dbsp4,
        P42 VALUES <= ('2008-12-31') IN dbsp4);
```

The following SPLIT PARTITION reports an error as it requires data movement. Not all existing rows will be in the same partition after split.

```
ALTER TABLE bar SPLIT PARTITION p3 INTO
       (P31 VALUES <= ('2007-06-30') IN dbsp3,
        P32 VALUES <= ('2007-12-31') IN dbsp3);
```

The following error is reported:

```
"No data move is allowed, cannot split partition p3."
```

The following SPLIT PARTITION reports an error, because it changes the partition boundary value.

```
ALTER TABLE bar SPLIT PARTITION p2 INTO
       (p21 VALUES <= ('2006-06-30') IN dbsp2,
        P22 VALUES <= ('2006-12-01') IN dbsp2);
```

The following error is reported:

```
"Boundary value for the partition p2 cannot be changed."
```

Merge partition p3 into p2. An error is reported as a merge from a higher boundary value partition into a lower boundary value partition is not allowed.

```
ALTER TABLE bar MERGE PARTITION p3 into p2;
```

The following error is reported:

```
"Partition 'p2' is not adjacent to or before partition
'p3'."
```

Merge partition p2 into p3:

```
ALTER TABLE bar MERGE PARTITION p2 INTO P3;
```

Rename partition p1 to p1_new:

```
ALTER TABLE bar RENAME PARTITION p1 TO p1_new;
```

Unpartition table bar:

```
ALTER TABLE bar UNPARTITION;
```

Partition table bar. This command reports an error, because all rows must be in the first partition.

```
ALTER TABLE bar PARTITION BY RANGE(c2)
    (p1 VALUES <= ('2005-12-31') IN dbsp1,
     P2 VALUES <= ('2006-12-31') IN DBSP2,
     P3 VALUES <= ('2007-12-31') IN dbsp3,
     P4 VALUES <= ('2008-12-31') IN dbsp4);
```

The following error is reported:

```
"All rows must be in the first partition."
```

Partition table bar:

```
ALTER TABLE bar PARTITION BY RANGE(c2)
    (p1 VALUES <= ('2008-12-31') IN dbsp1,
     P2 VALUES <= ('2009-12-31') IN dbsp2,
     P3 VALUES <= ('2010-12-31') IN dbsp3,
     P4 VALUES <= ('2011-12-31') IN dbsp4);
```

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 135.

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 135 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.

ALTER column-name column-alteration 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.
- **[CONSTRAINT constraint-name] CHECK (condition)** Use this clause to add a CHECK constraint on the column.

- **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.

DROP partition clause The DROP partition clause drops the specified partition. The rows are deleted and the partition definition is dropped. You cannot drop the last partition because dropping the last partition would transform a partitioned table to a non-partitioned table. (To merge a partitioned table, use UNPARTITION clause instead.) For example:

```
CREATE TABLE foo (c1 INT, c2 INT)
   PARTITION BY RANGE (c1)
   (P1 VALUES <= (100) IN dbsp1,
    P2 VALUES <= (200) IN dbsp2,
    P3 VALUES <= (MAX) IN dbsp3
   ) IN dbsp4);
LOAD TABLE ….
ALTER TABLE DROP PARTITION P1;
```

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 135 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.

DROP column-name Drop 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.

DROP CHECK Drop all check constraints for the table. This includes both table check constraints and column check constraints.

DROP CONSTRAINT constraint-name Drop the named constraint for the table or specified column.

DROP UNIQUE (column-name,…) Drop the unique constraints on the specified column(s). Any foreign keys referencing the unique 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.

DROP PRIMARY KEY Drop the primary key. 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.

DROP FOREIGN KEY role-name Drop 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.

DROP PARTITION The DROP PARTITION request deletes rows in partition P1 and drops the partition definition of P1. If a new row with value 99 for column c1 is inserted, it will be placed under partition p2 in dbspace dbsp2.

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.

ALTER Column MOVE TO The ALTER Column MOVE TO clause moves the specified column to the new dbspace for a non-partitioned table. The ALTER Column MOVE TO clause cannot be requested on a partitioned table. The ALTER Column MOVE PARTITION clause moves the column of the specified partition to the specified dbspace.

MOVE PARTITION The MOVE PARTITION clause moves the specified partition to the new dbspace.

MOVE TO The MOVE TO clause moves all table objects including columns, indexes, unique constraints, primary key, foreign keys, and metadata resided in the same dbspace as the table is mapped to the new dbspace.

Each table object can reside in only one dbspace. Any type of ALTER MOVE blocks any modification to the table for the entire duration of the move.

MOVE TABLE METADATA The MOVE TABLE METADATA clause moves the metadata of the table, such as the EBM, DeleteBM, and InsertBM of the table, to a new dbspace. For a partitioned table, the MOVE TABLE METADATA clause also moves metadata that is shared among partitions.

You must have DBA authority or have CREATE privilege on the new dbspace and be the table owner or have alter permission on the table.

SPLIT PARTITION The SPLIT PARTITION clause splits the specified partition into two partitions. In Sybase IQ 15.1, a partition can be split only if no data must be moved. All existing rows of the partition to be split must remain in a single partition after the split. The boundary value for *partition-decl-1* must be less than the boundary value of *partition-name* and the boundary value for *partition-decl-2* must be equal to the boundary value of *partition-name*. You can specify different names for the two new partitions. The old *partition-name* can only be used for the second partition, if a new name is not specified.

MERGE PARTITION The MERGE PARTITION clause merges *partition-name-1* into *partition-name-2*. In Sybase IQ 15.1, two partitions can be merged if they are adjacent partitions and the data resides on the same dbspace. You can only merge a partition with a lower partition value into the adjacent partition with a higher partition value. Note that the server does not check CREATE permission on the dbspace into which the partition is merged. For an example of how to create adjacent partitions, see Example 3 in CREATE TABLE statement.

UNPARTITION The UNPARTITION keyword removes partitions from a partitioned table. Each column is placed in a single dbspace. Note that the server does not check CREATE permission on the dbspace to which data of all partitions is moved. ALTER TABLE UNPARTITION blocks all database activities.

PARTITION BY The PARTITION BY clause partitions a non-partitioned table. In Sybase IQ 15.1, a non-partitioned table can be partitioned, if all existing rows belong to the first partition. You can specify a different dbspace for the first partition than the dbspace of the column or table. But existing rows are not moved. Instead, the proper dbspace for the column/partition is kept in SYS.ISYSIQPARTITIONCOLUMN for existing columns. Only the default or max identity column(s) that are added later for the first partition are stored in the specified dbspace for the first partition.

RENAME PARTITION The RENAME PARTITION clause renames an existing partition name to a new partition name.

Side effects

- Automatic commit. The ALTER and DROP 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.
- **Sybase** Some clauses are supported by Adaptive Server Enterprise.

Permissions

Must have DBA authority or CREATE permission on the new dbspace and be the table owner or have ALTER permission on the table. Requires exclusive access to the table.

See also

CREATE TABLE statement on page 135

DROP statement on page 177

“IDENTITY_INSERT option” on page 394

Chapter 3, “SQL Data Types” in *Reference: Building Blocks, Tables, and Procedures*

ALTER USER statement

Description

Changes user settings.

Syntax

Syntax 1

ALTER USER *user-name* [IDENTIFIED BY *password*] [LOGIN POLICY *policy-name*] [FORCE PASSWORD CHANGE { ON | OFF }]

Syntax 2

ALTER USER *user-name* [RESET LOGIN POLICY]

Examples

The following alters a user named SQLTester. The password is set to “welcome”. The SQLTester user is assigned to the Test1 login policy and the password does not expire on the next login.

```
ALTER USER SQLTester IDENTIFIED BY welcome
LOGIN POLICY Test1
FORCE PASSWORD CHANGE off;
```

Usage

user-name The name of the user.

IDENTIFIED BY clause Clause providing the password for the user.

policy-name The name of the login policy to assign the user. No change is made if the LOGIN POLICY clause is not specified.

FORCE PASSWORD CHANGE clause Controls whether the user must specify a new password when they log in. This setting overrides the password_expiry_on_next_login option setting in their policy.

RESET LOGIN POLICY clause Reverts the settings of the user's login to the original values in the login policy. This usually clears all locks that are implicitly set due to the user exceeding the failed logins or exceeding the maximum number of days since the last login. When you reset a login policy, a user can access an account that has been locked for exceeding a login policy option limit such as max_failed_login_attempts or max_days_since_login.

Enhanced ALTER LOGIN POLICY syntax for multiplex is described in *Using Sybase IQ Multiplex*.

User IDs and passwords cannot:

- Begin with white space, single quotes, or double quotes
- End with white space
- Contain semicolons

If you set the PASSWORD_EXPIRY_ON_NEXT_LOGIN value to ON, the passwords of all users assigned to this login policy expire immediately when they next log in. You can use the ALTER USER and LOGIN POLICY clauses to force a user to change the password when he next logs in.

Standards

- **SQL2003** Vendor extension.
- **Sybase** Not supported by Adaptive Server Enterprise.

Permissions

Must be owner of the view or have DBA authority.

See also

“ALTER LOGIN POLICY statement” on page 19

“COMMENT statement” on page 61

“CREATE LOGIN POLICY statement” on page 117

“CREATE USER statement” on page 151

“DROP LOGIN POLICY statement” on page 183

“DROP USER statement” on page 185

“GRANT statement” on page 206

“Managing login policies overview” in *SQL Anywhere Server – Database Administration > Configuring Your Database > Managing user IDs, authorities, and permissions*

“REVOKE statement” on page 287

ALTER VIEW statement

Description

Replaces a view definition with a modified version.

Syntax

Syntax 1

ALTER VIEW
… [*owner*.]*view-name* [(*column-name* [, …])]
… **AS** *select-statement*
… [**WITH CHECK OPTION**]

Syntax 2

ALTER VIEW
… [*owner*.]*view-name*
… { **SET HIDDEN** | **RECOMPILE** | **DISABLE** | **ENABLE** }

Usage

AS Purpose and syntax Identical to CREATE VIEW statement. See “CREATE VIEW statement” on page 155

WITH CHECK OPTION Purpose and syntax Identical to CREATE VIEW statement. See “CREATE VIEW statement” on page 155

SET HIDDEN Use the SET HIDDEN clause to obfuscate the definition of the view and cause the view to become hidden from view, for example in Sybase Central. Explicit references to the view still works.

Warning! The SET HIDDEN operation is irreversible.

RECOMPILE Recreates the column definitions for the view. Identical in functionality to the ENABLE clause, except you can use it on a view that is not disabled.

DISABLE Disables the view from use by the database server.

ENABLE Enables a disabled view, which causes the database server to recreate the column definitions for the view. Before you enable a view, you must enable any views on which it depends.

When you alter a view, existing permissions on the view are maintained and do not require reassignment. Instead of using the ALTER VIEW statement, you could also drop the view and recreate it using DROP VIEW and CREATE VIEW, respectively. If you do this, view permissions must be reassigned.

After completing the view alteration using Syntax 1, the database server recompiles the view. Depending on the type of change you made, if there are dependent views, the database server attempts to recompile them. If you made changes that impact a dependent view, you might need to alter the definition for the dependent view, as well. For more information about view alterations and how they impact view dependencies, see “View dependencies” in *SQL Anywhere Server – SQL Usage > Creating Databases > Working with database objects > Working with views*.

Warning! If the SELECT statement defining the view contains an asterisk (*), the number of the columns in the view could change if columns were added or deleted from the underlying tables. The names and data types of the view columns could also change.

Syntax 1 Alters the structure of the view. Unlike altering tables, where your change might be limited to individual columns, altering the structure of a view requires that you replace the entire view definition with a new definition, much as you would when creating the view. For a description of the parameters used to define the structure of a view, see “CREATE VIEW statement” on page 155.

Syntax 2 Changes attributes for the view, such as whether the view definition is hidden.

When you use SET HIDDEN, you can unload and reload the view into other databases. Debugging using the debugger does not show the view definition, nor is it available through procedure profiling. If you need to change the definition of a hidden view, you must drop the view and create it again using the CREATE VIEW statement.

When you use the DISABLE clause, the view is no longer available for use by the database server to answer queries. Disabling a view is similar to dropping one, except that the view definition remains in the database. Disabling a view also disables any dependent views. Therefore, the DISABLE clause requires exclusive access, not only to the view being disabled, but to any dependent views, which are also disabled.

Side Effects

Automatic commit.

All procedures and triggers are unloaded from memory, so that any procedure or trigger that references the view reflects the new view definition. The unloading and loading of procedures and triggers can have a performance impact if you regularly alter views.

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 155

DROP statement on page 177

“sa_dependent_views procedure,” Chapter 7, “System Procedures,” in *Reference: Building Blocks, Tables, and Procedures*

“Hiding the contents of procedures, functions, and views” in Chapter 1, “Using Procedures and Batches” in *System Administration Guide: Volume 2*

“View dependencies” in *SQL Anywhere Server – SQL Usage > Creating Databases > Working with database objects > Working with views*

BACKUP statement

Description

Backs up a Sybase IQ database on one or more archive devices.

Syntax

BACKUP DATABASE
[*backup-option*…]
TO *archive_device* [*archive-option*...]
… [**WITH COMMENT** *string*]

Parameters

backup-option:

{ **READWRITE FILES ONLY |**
READONLY *dbspace-or-file* [, …] }
CRC { **ON** | **OFF** }
ATTENDED { **ON** | **OFF** }
BLOCK FACTOR *integer*
{ **FULL** | **INCREMENTAL** | **INCREMENTAL SINCE FULL** }
VIRTUAL { **DECOUPLED |**
ENCAPSULATED *'shell_command'* }
WITH COMMENT *comment*

dbspace-or-file:

{ **DBSPACES** *identifier-list* | **FILES** *identifier-list* }

identifier-list:
identifier [, …]

archive-option:
SIZE *integer*
STACKER *integer*

Examples

Example 1 The following UNIX example backs up the iqdemo 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.

Example 2 The following BACKUP commands specify read-only files and dbspaces:

```
BACKUP DATABASE READONLY DBSPACES dsp1
TO '/dev/rmt/0'
```

```
BACKUP DATABASE READONLY FILES dsp1_f1, dsp1_f2
TO 'bkp.f1f2'
```

```
BACKUP DATABASE READONLY DBSPACES dsp2, dsp3
READONLY FILES dsp4_f1, dsp5_f2
TO 'bkp.RO'
```

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.

To make a virtual backup of one or more read-only dbspaces you may simply perform an OS level copy of the tablespaces, but Sybase recommends that you use the virtual backup statement because it records the backup in the IQ system tables. See “SYSIQBACKUPHISTORY system view” and “SYSIQBACKUPHISTORYDETAIL system view” in Chapter 8, “System Views” of *Reference: Building Blocks, Tables, and Procedures*.

READWRITE FILES ONLY may be used with FULL, INCREMENTAL, and INCREMENTAL SINCE FULL to restrict the backup to only the set of read-write files in the database. The read-only dbspaces/files must be IQ dbspaces.

If READWRITE FILES ONLY is used with an INCREMENTAL or INCREMENTAL SINCE FULL backup, the backup will *not* back up data on read-only dbspaces or dbfiles that has changed since the depends-on backup. If READWRITE FILES ONLY is not specified for an INCREMENTAL or INCREMENTAL SINCE FULL backup, the backup backs up all database pages that have changed since the depends-on backup, both on read-write and read-only dbspaces.

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.

The keyword INCREMENTAL is not allowed with READONLY FILES.

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 *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 *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 1-1: 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 Specifies 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 *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 a SQL Anywhere database, see “Backup and Data Recovery” in *SQL Anywhere Server – Database Administration > Maintaining Your Database* 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 279

Chapter 12, “Data Backup, Recovery, and Archiving,” in *System Administration Guide: Volume 1*

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 CompanyName, CAST(
                sum(SalesOrderItems.Quantity *
                Products.UnitPrice) AS INTEGER) VALUE
        FROM Customers
                LEFT OUTER JOIN Salesorders
                LEFT OUTER JOIN SalesOrderItems
                LEFT OUTER JOIN Products
        GROUP BY CompanyName ;
    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.

ATOMIC clause An atomic statement is a statement executed completely or not at all. For example, an UPDATE statement that updates thousands of rows might encounter an error after updating many rows. If the statement does not complete, all changes revert back to their original state. Similarly, if you specify that the BEGIN statement is atomic, the statement is executed either in its entirety or not at all.

For a complete description of compound statements and exception handling, see Chapter 1, “Using Procedures and Batches” in the *System Administration Guide: Volume 2*.

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 167

DECLARE CURSOR statement [ESQL] [SP] on page 159

LEAVE statement on page 229

RESIGNAL statement on page 278

SIGNAL statement on page 312

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 105.

Permissions

None

See also

CREATE INDEX statement on page 105

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 1-2 is a timeline of the commands executed within the two connections:

Table 1-2: first case command timeline

Conn1	Conn2
CONNECT	CONNECT
INSERT INTO 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 1-3: second command timeline

Conn1	Conn2
CONNECT	CONNECT
…	BEGIN TRANSACTION
INSERT INTO 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 360.

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 402

COMMIT statement on page 62

ROLLBACK statement on page 289

SAVEPOINT statement on page 291

CALL statement

Description

Invokes a procedure.

Syntax

Syntax 1

[*variable* =] **CALL** *procedure-name* ([*expression*] [, …])

Syntax 2

[*variable* =] **CALL** *procedure-name* ([*parameter-name* = *expression*] [, …])

Examples

Example 1 This example calls the sp_customer_list procedure. This procedure has no parameters, and returns a result set:

```
CALL sp_customer_list()
```

Example 2 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 CustomerID INT, OUT
Orders INT)
BEGIN
SELECT COUNT("DBA".SalesOrders.ID)
INTO Orders
FROM "DBA".Customers
KEY LEFT OUTER JOIN "DBA".SalesOrders
WHERE "DBA".Customers.ID = CustomerID ;
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 1, "Using Procedures and Batches" in the *System Administration Guide: Volume 2*.

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 186.

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 120

GRANT statement on page 206

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**

Examples

This procedure using a CASE statement classifies the products listed in the Products table of the demo 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"."Products"
   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.

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 SQL Anywhere, see the CASE statement Syntax 2 in "CASE statement" in *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (A-D)*.

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" in Chapter 2, "SQL Language Elements" in *Reference: Building Blocks, Tables, and Procedures*.

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 47

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 12, "Data Backup, Recovery, and Archiving," in the *System Administration Guide: Volume 1*.

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

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 192

CLOSE statement [ESQL] [SP]

Description

Closes a cursor.

Syntax

CLOSE *cursor-name*

Parameters

cursor-name:
{ *identifier* | *host-variable* }

Examples

Example 1 Close cursors in Embedded SQL:

```
EXEC SQL CLOSE employee_cursor;
EXEC SQL CLOSE :cursor_var;
```

Example 2 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 CompanyName,
            CAST(    sum(SalesOrderItems.Quantity *
            Products.UnitPrice) AS INTEGER) VALUE
        FROM Customers
        LEFT OUTER JOIN SalesOrders
        LEFT OUTER JOIN SalesOrderItems
        LEFT OUTER JOIN Products
    GROUP BY CompanyName ;
    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 159

OPEN statement [ESQL] [SP] on page 260

PREPARE statement [ESQL] on page 268

COMMENT statement

Description
Stores a comment in the system tables for a database object.

Syntax

COMMENT ON
{ **COLUMN** [*owner*.]*table-name*.*column-name*
| **DBSPACE** *dbspace-name*
| **EVENT** *event-name*
| **EXTERNAL ENVIRONMENT** *environment-name*
| **EXTERNAL OBJECT** *object-name*
| **FOREIGN KEY** [*owner*.]*table-name*.*role-name*
| **INDEX** [[*owner*.]*table*.]*index-name*
| **INTEGRATED LOGIN** *integrated-login-id*
| **JAVA CLASS** *java-class-name*
| **JAVA JAR** *java-jar-name*
| **KERBEROS LOGIN** "*client-Kerberos-principal*"
| **LOGIN POLICY** *policy-name*
| **MATERIALIZED VIEW** [*owner*.]*materialized-view-name*
| **PROCEDURE** [*owner*.]*table-name*
| **SERVICE** *web-service-name*
| **TABLE** [*owner*.]*table-name*
| **TRIGGER** [[*owner*.]*table-name*.]*trigger-name*
| **USER** *userid*
| **VIEW** [*owner*.]*view-name* }
IS *comment*

Parameters

comment:
{ *string* | NULL }

environment-name:
JAVA
| PERL
| PHP
| CLR
| C_ESQL32
| C_ESQL64
| C_ODBC32
| C_ODBC64

Examples

Example 1 Adds a comment to the Employees table:

```
COMMENT
ON TABLE Employees
IS "Employee information"
```

Example 2 Removes the comment from the Employees table:

```
COMMENT
ON TABLE Employees
IS NULL
```

Usage

The COMMENT statement allows you to set a remark (comment) for an object in the database. The COMMENT statement updates remarks listed in the ISYSREMARK system table. You can remove a comment by setting it to NULL. For a comment on an index or trigger, the owner of the comment is the owner of the table on which the index or trigger is defined.

The COMMENT ON DBSPACE, COMMENT ON JAVA JAR, and COMMENT ON JAVA CLASS statements allow you to set the Remarks column in the SYS.ISYSREMARK system table. The comment can be removed by setting it to NULL.

You cannot add comments for local temporary tables.

The *environment-name* is one of JAVA, PERL, PHP, CLR, C_ESQL32, C_ESQL64, C_ODBC32, or C_ODBC64.

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. (You must have DBA authority to issue this statement with the DBSPACE clause.)

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

Example 1 This statement commits the current transaction:

```
COMMIT
```

Example 2 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 369, "COOPERATIVE_COMMITS option" on page 369, "DELAYED_COMMITS option" on page 382, and "DELAYED_COMMIT_TIMEOUT option" on page 382. 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 51

CONNECT statement [ESQL] [DBISQL] on page 65

DISCONNECT statement [DBISQL] on page 176

ROLLBACK statement on page 289

SAVEPOINT statement on page 291

SET CONNECTION statement [DBISQL] [ESQL] on page 306

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 307

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**]

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, string, or host-variable

connect-string:
a valid connection string or host-variable

Examples

Example 1 This is an example 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";
```

Example 2 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 <machine>_iqdemo
USER "DBA"
IDENTIFIED BY sql
```

where *<machine>_iqdemo* is the engine name.

- Connect to the sample database using a connect string, from dbisql:

```
CONNECT
USING 'UID=DBA;PWD=sql;DBN=iqdemo'
```

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 *System Administration Guide: Volume 1*.

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 176

GRANT statement on page 206

SET CONNECTION statement [DBISQL] [ESQL] on page 306

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*[(*collation-tailoring-string*)]]
… [**ENCRYPTED** [**TABLE**] {*algorithm-key-spec* | **OFF** }]
… { … [**BLANK PADDING ON**]
… [**JAVA** { **ON** | **OFF** }]
… [**JCONNECT** { **ON** | **OFF** }]
… [**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

collation-tailoring-string:
keyword=value

algorithm-key-spec:
ON
| [ON] KEY *key* [ALGORITHM *AES-algorithm*]
| [ON] ALGORITHM *AES-algorithm* KEY *key*
| [ON] ALGORITHM 'SIMPLE'

AES-algorithm:
'AES' | 'AES256' | 'AES_FIPS' | 'AES256_FIPS'

key:
quoted string

Examples

Example 1 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
```

Example 2 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/bigdb.iqmsg'
TEMPORARY PATH '/dev/md/rdsk/bigtmp'
```

Example 3 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
```

Example 4 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
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 SQL 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 SQL 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.

  IQ_SYSTEM_MAIN is a special dbspace that contains all structures necessary for the database to open: the IQ db_identity blocks, the IQ checkpoint log, the IQ rollforward/rollback bitmaps of each committed transaction and each active checkpointed transaction, the incremental backup bitmaps, and the freelist root pages. IQ_SYSTEM_MAIN is always online when the database is open.

  The administrator can allow user tables to be created in IQ_SYSTEM_MAIN, especially if these tables are small, very important tables. However, the more common case is that immediately after creating the database, the administrator creates a second main dbspace, revokes CREATE privilege IN DBSPACE IQ_SYSTEM_MAIN from all users, grants CREATE IN DBSPACE for the new main dbspace to selected users, and sets PUBLIC.default_dbspace to the new main dbspace.

- *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.

- *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.

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 To create multiplex databases, see *Using Sybase IQ Multiplex*.

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 operating system 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 insensitive for databases created with CASE IGNORE or CASE RESPECT.

The default (RESPECT) is that all comparisons are case sensitive. CASE RESPECT provides better performance than CASE IGNORE.

Character strings inserted into tables are always stored in the case they are entered, regardless of whether the database is case sensitive or not. If the string Value is inserted into a character data type column, the string is always stored 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. The IQ server may return results in any combination of lowercase and uppercase, so you cannot expect case sensitive results in a database that is case insensitive (CASE IGNORE).

For example, given the following table and data:

```
CREATE TABLE tb (id int NOT NULL,
                 string VARCHAR(30) NOT NULL);
INSERT INTO tb VALUES (1, 'ONE');
SELECT * FROM tb WHERE string = 'oNe';
```

The result of the SELECT can be ‘oNe’ (as specified in the WHERE clause) and not necessarily ‘ONE’ (as stored in the database).

Similarly, the result of

```
SELECT * FROM tb WHERE string = 'One';
```

can be ‘One’ and the result of

```
SELECT * FROM tb WHERE string = 'ONe';
```

can be ‘ONe’.

All databases are created with at least one user ID:

```
DBA
```

and password:

```
sql
```

In new databases, all passwords are case sensitive, regardless of the case-sensitivity of the database. The user ID is unaffected by the CASE RESPECT setting.

PAGE SIZE The page size for the SQL Anywhere segment of the database (containing the catalog tables) can be 4096, 8192, 16384, or 32768 bytes, with 4096 being the default. Normally, you should use the default, 4096 (4KB). Large databases might need a larger page size than the default and could see performance benefits as a result. 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. A larger page size might also be needed to execute queries involving large numbers of tables or views.

Because the default Catalog page size is 4KB, this is only a problem when the connection is to a database such as utility_db, which has a page size of 1024. This restriction may cause RESTORE commands that reference numerous dbspaces to fail. To avoid the problem, make sure the length of SQL command lines is less than the Catalog page size.

Alternatively, start the engine with `-gp 32768` to increase Catalog page size.

COLLATION The collation sequence used for sorting and comparison of character data types in the database. The collation provides character comparison and ordering information for the encoding (character set) being used. If the COLLATION clause is not specified, Sybase IQ chooses a collation based on the operating system language and encoding.

For most operating systems, 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’).

The collation can be chosen from a list of supported collations. For SQL Anywhere databases created on a Sybase IQ server, the collation can also be the Unicode Collation Algorithm (UCA). If UCA is specified, you should also specify the ENCODING clause. For more information on the ENCODING clause, see “CREATE DATABASE statement” in *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (A-D)*.

Sybase IQ does not support any of the UCA based collations for IQ databases. If a UCA based collation is specified in the CREATE DATABASE statement for an IQ database, the server returns the error `"UCA collation is not supported"` and database creation fails.

Carefully choosing your collation is important. The collation cannot be changed after the database is created. For information on choosing a collation, see Chapter 11, “International Languages and Character Sets” in the *System Administration Guide: Volume 1*.

Optionally, you can specify collation tailoring options (*collation-tailoring-string*) for additional control over the sorting and comparing of characters. These options take the form of keyword=value pairs, assembled in parentheses, following the collation name.

Note Several collation tailoring options are supported when specifying the UCA collation for a SQL Anywhere database created on an Sybase IQ server. For all other collations and for Sybase IQ, only case sensitivity tailoring is supported. Also, databases created with collation tailoring options cannot be started using a pre-15.0 database sever.

Table 1-4 contains the supported keyword, allowed alternate forms, and allowed values for the collation tailoring option (*collation-tailoring-string*) for a Sybase IQ database.

Table 1-4: Collation tailoring option for Sybase IQ

Keyword	Collation	Alternate forms	Allowed values
CaseSensitivity	All supported collations	CaseSensitive, Case	• respect Respect case differences between letters. For the UCA collation, this is equivalent to UpperFirst. For other collations, it depends on the collation itself. • ignore Ignore case differences between letters. • UpperFirst Always sort upper case first (Aa). • LowerFirst Always sort lowercase first (aA).

For syntax and a complete list of the collation tailoring options supported when specifying the UCA collation for a SQL Anywhere database, see "CREATE DATABASE statement" in *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (A-D)*.

ENCRYPTED Encryption makes the data stored in your physical database file unreadable. Use the CREATE DATABASE ENCRYPTED keyword without the TABLE keyword when you want to encrypt the entire database. Use the ENCRYPTED TABLE clause when you only want to enable table encryption for SQL Anywhere tables. Table level encryption is not supported for Sybase IQ tables. Enabling table encryption means that the tables that are subsequently created or altered using the ENCRYPTED clause are encrypted using the settings you specified at database creation.

There are two levels of database and table encryption: simple and strong.

- Simple encryption is equivalent to obfuscation. The data is unreadable, but someone with cryptographic expertise could decipher the data. For simple encryption, specify the CREATE DATABASE clause ENCRYPTED ON ALGORITHM ‘SIMPLE’, ENCRYPTED ALGORITHM ‘SIMPLE’, or specify the ENCRYPTED ON clause without specifying an algorithm or key.
- 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. For strong encryption, specify the CREATE DATABASE clause ENCRYPTED ON ALGORITHM with a 128-bit or 256-bit AES algorithm and use the KEY clause to specify an encryption key. You should 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.

  This encryption key is required each time you start the database.

Warning! Protect your encryption 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.

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.

If the ENCRYPTED clause is used but no algorithm is specified, the default is AES. Encryption is OFF by default.

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.

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.

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.

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 *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).

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 by not specifying IQ SIZE, 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 1-5 up to a maximum of 128GB. The default for operating system files depends on IQ PAGE SIZE:

Table 1-5: 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 The 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 4, “Managing System Resources” in the *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 dbfile 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 dbfile 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 *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 by not specifying TEMPORARY SIZE, 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 81

DROP DATABASE statement on page 181

"CREATE DATABASE statement" in *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (A-D)*

Chapter 11, "International Languages and Character Sets" in the *System Administration Guide: Volume 1*

CREATE DBSPACE statement

Description

Creates a new dbspace and the associated dbfiles for the IQ main store or catalog store.

Syntax

Syntax 1

Use for catalog store dbspaces only (SA dbspaces).

CREATE DBSPACE *dbspace-name* **AS** *file-path* **CATALOG STORE**

Syntax 2

Use for IQ dbspaces.

CREATE DBSPACE *dbspace-name* **USING** *file-specification*
[**IQ STORE**] *iq-dbspace-opts*

Parameters

file-specification:
{ *single-path-spec* | *new-file-spec* [, ...] }

single-path-spec:
'file-path' | *iq-file-opts*

new-file-spec:
FILE *logical-file-name* | *'file-path' iq-file-opts*

iq-file-opts:
[[SIZE] *file-size*]
…[KB | MB | GB | TB]]
[RESERVE *size*
…[KB | MB | GB | TB]]

iq-dbspace-opts:
[STRIPING] {ON | OFF}]
…[STRIPESIZEKB *sizeKB*]

Examples

Example 1 Creates a dbspace called *DspHist* for the IQ main store with two files on a UNIX system. Each file has 1GB and can grow 500MB:

```
CREATE DBSPACE DspHist USING FILE
FileHist1 '/History1/data/file1'
SIZE 1000 RESERVE 500,
FILE FileHist2 '/History1/data/file2'
SIZE 1000 RESERVE 500;
```

Example 2 Creates a second catalog dbspace called *DspCat2*:

```
CREATE DBSPACE DspCat2 AS
'catalog_file2'
CATALOG STORE;
```

Example 3 Creates an IQ main dbspace called *EmpStore1* for the IQ store (three alternate syntax examples):

```
CREATE DBSPACE EmpStore1
USING FILE EmpStore1
'EmpStore1.IQ' SIZE 8 MB IQ STORE;
```

```
CREATE DBSPACE EmpStore1
USING FILE EmpStore1
'EmpStore1.IQ' 8 IQ STORE;
```

```
CREATE DBSPACE EmpStore1
USING FILE EmpStore1
'EmpStore1.IQ' 8;
```

Usage

CREATE DBSPACE creates a new dbspace for the IQ main store or the catalog 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.

Syntax 1 creates a dbspace for the catalog store, where both dbspace and dbfile have the same logical name. Each dbspace in the catalog store has a single file.

new-file-spec creates a dbspace for the IQ main store. One or more dbfiles can be specified for the IQ main store. The dbfile name and physical file path are required for each file and must be unique.

The dbspace name and dbfile names are always case insensitive. The physical file paths have the case sensitivity of the operating system, if the database is CASE RESPECT and are case insensitive, if the database is CASE IGNORE.

You cannot create a dbspace for an IQ temporary store. A single temporary dbspace, IQ_SYSTEM_TEMP, is created when you create a new database or upgrade one created prior to Sybase IQ 15.1. You can add additional files to the IQ_SYSTEM_TEMP dbspace using the ALTER DBSPACE ADD FILE syntax.

If you do not specify striping or stripe size, the default values of the options DEFAULT_DISK_STRIPING and DEFAULT_KB_PER_STRIPE apply.

RESERVE clause Specifies the size in kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB) of space to reserve, so that the dbspace can be increased in size in the future. The size parameter can be any number greater than 0; megabytes is the default. The reserve cannot be changed after the dbspace dbfile 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.

See CREATE DATABASE statement on page 68 for the names and types of files created by default.

Note For information on creating dbspaces for a multiplex database, see *Using Sybase IQ Multiplex*.

You can create a unique path in any of these ways:

- Specify a different extension for each file (for example, *mydb.iq*)
- Specify a different file name (for example, *mydb2.iq*)
- Specify a different path name (for example, */iqfiles/main/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 operating system file.

The *dbspace-name* and *dbfile-name* are internal names for dbspaces and dbfiles. The *filepath* is the actual operating system file name of the dbfile, with a preceding path where necessary. A *filepath* without an explicit directory is created in the same directory as the catalog store of the database. Any relative directory is relative to the catalog store.

SIZE clause Specifies the size, from 0 to 4 terabytes, of the operating system file you specify in *filepath*. 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. You cannot specify the SIZE clause for the catalog store.

A SIZE value of 0 creates a dbspace of minimum size, which is 8Mb for IQ main 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.

A database can have up to (32k - 1) 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 177

Chapter 5, “Working with Database Objects,” in the *System Administration Guide: Volume 1*

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 *System Administration Guide: Volume 1*.

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 177

Chapter 3, "SQL Data Types" in *Reference: Building Blocks, Tables, and Procedures*

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
| IQMainDBSpaceFree
| IQTempDBSpaceFree | 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

Example 1 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
```

Example 2 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 6, “Automating Tasks Using Schedules and Events” in the *System Administration Guide: Volume 2*.

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 want 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,” in *System Administration Guide: Volume 1*.

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] in Chapter 4, “SQL Functions” in *Reference: Building Blocks, Tables, and Procedures*.

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 14

BEGIN … END statement on page 47

COMMENT statement on page 61

DROP statement on page 177

TRIGGER EVENT statement on page 319

Chapter 6, "Automating Tasks Using Schedules and Events" in the *System Administration Guide: Volume 2*

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

Example 1 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'
```

Example 2 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'
```

Example 3 This example creates a proxy table named rda_employee for the Employees table at the Sybase IQ remote server iqdemo:

```
CREATE EXISTING TABLE rda_employee
AT 'iqdemo..dba.Employees'
```

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_a1 to the SQL Anywhere database mydb on the remote server myasa:

```
CREATE EXISTING TABLE
proxy_a1
AT 'myasa;mydb;;a1'
```

In a simplex environment, creating a proxy table which refers to a remote table on the same node is not allowed. In a multiplex environment, creating a proxy table which refers to the remote table defined within the multiplex is not allowed.

For example, in a simplex environment, if you try to create proxy table proxy_e which refers to base table Employees defined on the same node, then the CREATE EXISTING TABLE statement is rejected with an error message. In a multiplex environment, the CREATE EXISTING TABLE statement is rejected if you create proxy table proxy_e from any node (coordinator or secondary) which refers to remote table Employees defined within a multiplex.

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 135

Chapter 4, "Accessing Remote Data" and Chapter 5, "Server Classes for Remote Data Access" in the *System Administration Guide: Volume 2*

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 ISYSSERVER system table. For more information, see CREATE SERVER statement on page 130.

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. If you omit the IDENTIFIED BY clause, the password is sent to the remote server as NULL. If you specify IDENTIFIED BY "''" (an empty string), then the password sent is the empty string.

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 users with DBA authority can add or modify an external login for *login-name*.

See also

DROP EXTERNLOGIN statement on page 182

INSERT statement on page 216

CREATE FUNCTION statement

Description

Creates a new function in the database.

Syntax

Syntax 1

CREATE [**TEMPORARY**] **FUNCTION** [*owner*.]*function-name*
([*parameter*, …])
RETURNS *data-type routine-characteristics*
[**SQL SECURITY** { **INVOKER** | **DEFINER** }]
{ *compound-statement*
| **AS** *tsql-compound-statement*
| *external-name* }

Syntax 2

CREATE FUNCTION [*owner*.]*function-name* ([*parameter*, …])
RETURNS *data-type*
URL *url-string*
[**HEADER** *header-string*]
[**SOAPHEADER** *soap-header-string*]
[**TYPE** { '**HTTP**[:{ **GET** | **POST** }] ' | '**SOAP**[:{ **RPC** | **DOC** }]' }]
[**NAMESPACE** *namespace-string*]
[**CERTIFICATE** *certificate-string*]
[**CLIENTPORT** *clientport-string*]
[**PROXY** *proxy-string*]

Parameters

url-string:
' { HTTP | HTTPS | HTTPS_FIPS }://[*user*:*password*@]*hostname*[:*port*][/*path*] '

parameter:
IN *parameter-name data-type* [DEFAULT *expression*]

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*; …'

operating-system:
UNIX

java-call:
'[*package-name*.]*class-name.method-name method-signature*'

method-signature:
([*field-descriptor*, ….]) *return-descriptor*

field-descriptor and *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 (givenname, surname)
FROM Employees
```

fullname (givenname, surname)
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

The CREATE FUNCTION statement 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 non-aggregate functions.

The following sections describe 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 be prefixed by the keyword IN, signifying that the argument is an expression that provides a value to the function.

When functions are executed, not all parameters need to be specified. If a default value is provided in the CREATE FUNCTION statement, missing parameters are assigned the default values. If an argument is not provided by the caller and no default is set, an error is given.

Specifying TEMPORARY (CREATE TEMPORARY FUNCTION) means that the function is visible only by the connection that created it, and that it is automatically dropped when the connection is dropped. Temporary functions can also be explicitly dropped. You cannot perform ALTER, GRANT, or REVOKE operations on them, and unlike other functions, temporary functions are not recorded in the catalog or transaction log.

Temporary functions execute with the permissions of their creator (current user), and can only be owned by their creator. Therefore, do not specify owner when creating a temporary function.

Temporary functions can be created and dropped when connected to a read-only database.

SQL SECURITY Defines whether the function is executed as the INVOKER, the user who is calling the function, or as the DEFINER, the user who owns the function. The default is DEFINER.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the procedure. Also, when SQL SECURITY INVOKER is specified, name resolution is done as the invoker as well. Therefore, take care to qualify all object names (tables, procedures, and so on) with their appropriate owner.

compound-statement A set of SQL statements bracketed by BEGIN and END, and separated by semicolons. See BEGIN … END statement on page 47.

tsql-compound-statement A batch of Transact-SQL statements. See “Transact-SQL batch overview” in Appendix A, “Compatibility with Other Sybase Databases” in *Reference: Building Blocks, Tables, and Procedures* and CREATE PROCEDURE statement [T-SQL] on page 127.

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.

The EXTERNAL NAME clause is not supported for temporary functions. For information about external library calls, see “Calling external libraries from procedures” in *SQL Anywhere Server – Programming > SQL Anywhere Data Access APIs > SQL Anywhere External Function API*.

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 120.

ON EXCEPTION RESUME Uses Transact-SQL-like error handling. See CREATE PROCEDURE statement on page 120.

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.

All user-defined functions are treated as deterministic unless they are declared NOT DETERMINISTIC. Deterministic functions return a consistent result for the same parameters and are free of side effects. That is, the database server assumes that two successive calls to the same function with the same parameters will return the same result without unwanted side-effects on the semantics of the query.

If a function returns a result set, it cannot also set output parameters or return a return value.

Note User-defined functions are processed by SQL Anywhere. They do not take advantage of the performance features of Sybase IQ. Queries that include user-defined functions run at least 10 times slower than queries without them.

In certain cases, differences in semantics between SQL Anywhere and Sybase IQ can produce different results for a query if it is issued in a user-defined function. For example, Sybase IQ treats the CHAR and VARCHAR data types as distinct and different, while 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 “ALTER FUNCTION statement” in *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (A-D)*.

URL For use only when defining an HTTP or SOAP web services client function. Specifies the URL of the web service. The optional user name and password parameters provide a means of supplying the credentials needed for HTTP basic authentication. HTTP basic authentication base-64 encodes the user and password information and passes it in the “Authentication” header of the HTTP request.

For web service client functions, the return type of SOAP and HTTP functions must one of the character data types, such as VARCHAR. The value returned is the body of the HTTP response. No HTTP header information is included. If more information is required, such as status information, use a procedure instead of a function.

Parameter values are passed as part of the request. The syntax used depends on the type of request. For HTTP:GET, the parameters are passed as part of the URL; for HTTP:POST requests, the values are placed in the body of the request. Parameters to SOAP requests are always bundled in the request body.

HEADER When creating HTTP web service client functions, use this clause to add or modify HTTP request header entries. Only printable ASCII characters can be specified for HTTP headers, and they are case-insensitive. For more information about how to use this clause, see the HEADER clause of the CREATE PROCEDURE statement on page 120.

For more information about using HTTP headers, see “Working with HTTP headers” in *SQL Anywhere Server – Programming > SQL Anywhere Data Access APIs > SQL Anywhere web services*.

SOAPHEADER When declaring a SOAP web service as a function, use this clause to specify one or more SOAP request header entries. A SOAP header can be declared as a static constant, or can be dynamically set using the parameter substitution mechanism (declaring IN, OUT, or INOUT parameters for hd1, hd2, and so on). A web service function can define one or more IN mode substitution parameters, but can not define an INOUT or OUT substitution parameter. For more information about how to use this clause, see the SOAPHEADER clause of the “CREATE PROCEDURE statement (web services)” in *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (A-D)*.

TYPE Specifies the format used when making the web service request. If SOAP is specified or no type clause is included, the default type SOAP:RPC is used. HTTP implies HTTP:POST. Since SOAP requests are always sent as XML documents, HTTP:POST is always used to send SOAP requests.

NAMESPACE Applies to SOAP client functions only and identifies the method namespace usually required for both SOAP:RPC and SOAP:DOC requests. The SOAP server handling the request uses this namespace to interpret the names of the entities in the SOAP request message body. The namespace can be obtained from the WSDL description of the SOAP service available from the web service server. The default value is the procedure's URL, up to but not including the optional path component.

CERTIFICATE To make a secure (HTTPS) request, a client must have access to the certificate used by the HTTPS server. The necessary information is specified in a string of semicolon-separated key/value pairs. The certificate can be placed in a file and the name of the file provided using the file key, or the whole certificate can be placed in a string, but not both. The following keys are available:

Key	Abbreviation	Description
file		File name of certificate
certificate	cert	The certificate
company	co	Company specified in the certificate
unit		Company unit specified in the certificate.
name		Common name specified in the certificate

Certificates are required only for requests that are either directed to an HTTPS server or can be redirected from a non-secure to a secure server.

CLIENTPORT Identifies the port number on which the HTTP client procedure communicates using TCP/IP. It is provided for and recommended only for connections across firewalls, as firewalls filter according to the TCP/UDP port. You can specify a single port number, ranges of port numbers, or a combination of both; for example, CLIENTPORT '85,90-97'.

See “ClientPort parameter [CPort],” Chapter 4, “Connection and Communication Parameters,” in *System Administration Guide: Volume 1*.

PROXY Specifies the URI of a proxy server. For use when the client must access the network through a proxy. Indicates that the procedure is to connect to the proxy server and send the request to the web service through it.

Side effects

Automatic commit.

Standards

- **SQL 2003** 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

“ALTER FUNCTION statement” on page 16

BEGIN … END statement on page 47

CREATE PROCEDURE statement on page 120

DROP statement on page 177

RETURN statement on page 286

Chapter 1, “Using Procedures and Batches” in the *System Administration Guide: Volume 2*

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

Example 1 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 )
```

Example 2 Creates a High_Group index on the ID column of the SalesOrderItems table. The data pages for this index are allocated from dbspace Dsp5.

```
CREATE
HG INDEX id_hg
ON SalesOrderItems
( ID ) IN Dsp5
```

Example 3 Creates a High_Group index on the SalesOrderItems table for the ProductID column:

```
CREATE HG INDEX item_prod_hg
ON Sales_OrderItems
( ProductID)
```

Example 4 Creates a Low_Fast index on the SalesOrderItems table for the same ProductID column without any notification messages:

```
CREATE LF INDEX item_prod
ON SalesOrderItems
( ProductID)
 NOTIFY 0
```

Example 5 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 SalesOrders table for the OrderDate column.

```
CREATE DTTM INDEX order_dttm
ON SalesOrders
( OrderDate )
```

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 *System Administration Guide: Volume 1* 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 200.) 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 109 for more information.

IN Specifies index placement. If the IN clause is omitted, the index is created in the dbspace where the table is created. 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. Sybase IQ ensures that any *dbspace-name* you specify is appropriate for the index. If you try to specify IQ_SYSTEM_MAIN or other main dbspaces for indexes on temporary tables, or vice versa, you receive an error. Dbspace names are case insensitive 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 1-6: 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 CompanyName 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 CompanyName
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 and WD index types can be created on LONG VARCHAR data. 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 makes 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, and 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 5KB. CHAR or VARCHAR data greater than 255 bytes are 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.

Queries with a single column in the ORDER BY clause run faster using multicolumn HG indexes. For example:

```
SELECT abs (x) from t1
ORDER BY x
```

In the above example, the HG index vertically projects *x* in sorted order.

To enhance query performance, use multicolumn HG indexes to run ORDER BY operations on more than one column (that can also include ROWID) in the SELECT or ORDER BY clause with the following conditions:

- All projected columns, plus all ordering columns (except ROWID), exist within the index
- The ordering keys match the leading HG columns, in order

If more than one multicolumn HG index satisfies these conditions, the index with the lowest distinct counts is used.

If a query has an ORDER BY clause, and the ORDER BY column list is a prefix of a multicolumn index where all columns referenced in the SELECT list are present in a multicolumn index, then the multicolumn index performs vertical projection; for example:

```
SELECT x,z,y FROM T
ORDER BY x,y
```

If expressions exist on base columns in the SELECT list, and all the columns referenced in all the expressions are present in the multicolumn index, then the query will use a multicolumn index; for example:

```
SELECT power(x,2), x+y, sin(z) FROM T
ORDER BY x,y
```

In addition to the two previous examples, if the ROWID() function is in the SELECT list expressions, multicolumn indexes will be used. For example:

```
SELECT rowid()+x, z FROM T
ORDER BY x,y,z
```

In addition to the three previous examples, if ROWID() is present at the end of an ORDER BY list, and if the columns of that list—except for ROWID()—use multicolumn indexes in the exact order, multicolumn indexes will be used for the query. For example:

```
SELECT z,y FROM T
ORDER BY x,y,z,ROWID()
```

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 values in a unique index column or columns 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 or columns 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 50 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.

Index names must be unique on a given table for both Sybase IQ and Adaptive Server Enterprise.

Permissions

Must have DBA authority or RESOURCE authority and CREATE privilege in the specified dbspace to create an index.

See also

CREATE JOIN INDEX statement on page 114

DROP statement on page 177

"INDEX_PREFERENCE option" on page 398

Chapter 6, "Using Sybase IQ Indexes," in the *System Administration Guide: Volume 1*

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*
IN *dbspace-name*

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

Example 1 Creates a join index between the Departments and Employees tables using the DepartmentID column, which is the primary key for Departments and foreign key for Employees.

```
CREATE JOIN INDEX emp_dept_join
FOR Departments FULL OUTER JOIN Employees
ON Departments.DepartmentID = Employees.DepartmentID
```

Example 2 Creates tables t1 and t2, where future data allocation is from the default dbspace, and join index t1t2, where future data allocation is from dbspace Dsp6.

```
CREATE TABLE t1(c1 int, c2 char(5));
CREATE TABLE t2(c1 int, c3 char(5));
CREATE JOIN INDEX t1t2 FOR t1
   FULL OUTER JOIN t2 ON t2.c1=t1.c1 IN Dsp6;
```

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.

IN Specifies the join index placement. If the IN clause is omitted, Sybase IQ creates the join index in the default dbspace (as specified by the option default_dbspace.)

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 data type, 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 *System Administration Guide: Volume 1*.

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, be the owner of all tables involved in the join, and have CREATE permission in the dbspace.

See also

CREATE INDEX statement on page 105

CREATE TABLE statement on page 135

Chapter 6, "Using Sybase IQ Indexes," in *System Administration Guide: Volume 1*

CREATE LOGIN POLICY statement

Description

Creates a login policy in the database.

Syntax

CREATE LOGIN POLICY *policy-name policy-options*

Parameters

policy-options:
policy-option [*policy-option...*]

policy_option:
policy-option-name =*policy-option-value policy-option-value*={ UNLIMITED | ROOT | *legal-option-value* }

Examples

The following example creates the *Test1* login policy. This example has an unlimited password life and allows the user a maximum of five attempts to enter a correct password before the account is locked.

```
CREATE LOGIN POLICY Test1
password_life_time=UNLIMITED
max_failed_login_attempts=5;
```

Usage

policy-name is the name of the login policy.

policy-option-name is the name of the login policy option. If you do not specify an option, the value from the root login policy is applied.

policy-option-value is the value assigned to the login policy option. If you specify UNLIMITED, no limits are imposed.

If you do not specify a policy option, values for the login policy are taken from the root login policy. Table 1-7 describes the default options for the root login policy.

Table 1-7: Login policy options

Option	Description	Values	Initial value for ROOT policy	Applies to
locked	If the value for this option is ON, users are prohibited from establishing new connections	ON, OFF	OFF	Users without DBA authority only
max_connections	The maximum number of concurrent connections allowed for a user.	0 - 2147483647	Unlimited	Users without DBA authority only
max_days_since_login	The maximum number of days that can elapse between two successive logins by the same user.	0 - 2147483647	Unlimited	Users without DBA authority only
max_failed_login_attempts	The maximum number of failed attempts, since the last successful attempt, to login to the user account before the account is locked.	0 - 2147483647	Unlimited	Users without DBA authority only
max_non_dba_connections	The maximum number of concurrent connections that a user without DBA authority can make. This option is only supported in the root login policy.	0 - 2147483647	Unlimited	Users without DBA authority only. Only to the root login policy.
password_expiry_on_next_login	If the value for this option is ON, the user's password will expire in the next login.	ON, OFF	OFF	All users including those with DBA authority
password_grace_time	The number of days before password expiration during which login is allowed but the default post_login procedure issues warnings.	0 - 2147483647	0	All users including those with DBA authority
password_life_time	The maximum number of days before a password must be changed.	0 - 2147483647	Unlimited	All users including those with DBA authority

Permissions Must have DBA authority.

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 270

RAISERROR statement [T-SQL] on page 274

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

Example 1 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"."Products"
    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
```

Example 2 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 CompanyName, CAST(
    sum(SalesOrderItems.Quantity *
    Products.UnitPrice) AS INTEGER) VALUE
    FROM Customers
    LEFT OUTER JOIN SalesOrdes
    LEFT OUTER JOIN SalesorderItems
    LEFT OUTER JOIN Products
    GROUP BY CompanyName ;


    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 47.

Note There are two ways to create stored procedures: SQL92 and T-SQL. BEGIN TRANSACTION, for example, is T-SQL specific when using CREATE PROCEDURE syntax. Do not mix syntax when creating stored procedures.

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 and must be prefixed by IN, OUT or INOUT. See Chapter 3, "SQL Data Types" in *Reference: Building Blocks, Tables, and Procedures.* 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 3, "SQL Data Types" in *Reference: Building Blocks, Tables, and Procedures*.

For more information on returning result sets from procedures, see Chapter 1, "Using Procedures and Batches" in the *System Administration Guide: Volume 2*.

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 GivenName
        FROM Employees
    ELSE
        SELECT Surname,GivenName
        FROM Employees
    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 427.

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.

Remote procedures can return only up to 254 characters in output variables.

For information on remote servers, see CREATE SERVER statement on page 130. For information on using remote procedures, see the section “Using remote procedure calls (RPCs)” in Chapter 4, “Accessing Remote Data” in the *System Administration Guide: Volume 2*.

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 1-8.

Table 1-8: 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-name* class. The class name must be fully qualified, and any dot in the name must be replaced by a backslash. 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," in Chapter 14, "Troubleshooting Hints," in the *System Administration Guide: Volume 1*.

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 47

CALL statement on page 55

DROP statement on page 177

EXECUTE IMMEDIATE statement [ESQL] [SP] on page 190

GRANT statement on page 206

"Copy Definition utility (defncopy)" in Chapter 3, "Database Administration Utilities" of the *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 Employees.Surname, Employees.givenName
   FROM Departmens, Employees
   WHERE Departments.DepartmentName = @deptname
   AND Departments.DepartmentID =
        Employees.DepartmentID
```

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 Employees.SurName, Employees.GivenName
   FROM Departments, Employees
   WHERE Departments.DepartmentName = deptname
   AND Departments.DepartmentID =
       Employees.DepartmentID
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.

Note There are two ways to create stored procedures: T-SQL and SQ/92. BEGIN TRANSACTION, for example, is T-SQL specific when using CREATE PROCEDURE syntax. Do not mix syntax when creating stored procedures.

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. SQL 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 120

"Copy Definition utility (defncopy)" in Chapter 3, "Database Administration Utilities" of the *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 135

CREATE VIEW statement on page 155

GRANT statement on page 206

CREATE SERVER statement

Description

Adds a server to the ISYSSERVER 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

Example 1 Creates a remote server for the JDBC-based Adaptive Server Enterprise server named ase_prod. Its machine name is “banana” and port number is 3025.

```
CREATE SERVER ase_prod
CLASS 'asejdbc'
USING 'banana:3025'
```

Example 2 Creates a SQL 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'
```

Example 3 Creates 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 5, “Server Classes for Remote Data Access” in the *System Administration Guide: Volume 2*.

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 SQL Anywhere is 2638.
- **dbname** For SQL 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 Chapter 5, “Server Classes for Remote Data Access” in the *System Administration Guide: Volume 2*.

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 DBA authority to execute this command.

See also

“ALTER SERVER statement” on page 20

“DROP SERVER statement” on page 183

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

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.ISYSTAB
```

After executing this statement, use any Web browser to open the URL *http://localhost/tables*.

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 23

"DROP SERVICE statement" on page 184

"Introduction to web services" in *SQL Anywhere Server – Programming > SQL Anywhere Data Access APIs > SQL Anywhere web services*

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** *dbspace-name*]
…[**ON COMMIT** { **DELETE** | **PRESERVE** } **ROWS**
| **NOT TRANSACTIONAL**]
[**AT** *location-string*]
[**PARTITION BY** *range-partitioning-scheme*]

Parameters

column-definition:
column-name data-type [[NOT] NULL]
[IN *dbspace-name*]
[DEFAULT *default-value* | IDENTITY]
[PARTITION (*partition-name* IN *dbspace-name* [, ...])]

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:
[CONSTRAINT *constraint-name*] {
{ UNIQUE
| PRIMARY KEY
| REFERENCES *table-name* [(*column-name*)] [*action*]
} [IN *dbspace-name*]
| CHECK (*condition*)
| IQ UNIQUE (*integer*) }

table-constraint:
[CONSTRAINT *constraint-name*] {
{ UNIQUE (*column-name* [, *column-name*] …)
| PRIMARY KEY (*column-name* [, *column-name*] …)
} [IN *dbspace-name*]
| *foreign-key-constraint*| CHECK (*condition*)
| IQ UNIQUE (*integer*) }

foreign-key-constraint:
FOREIGN KEY [*role-name*]
[(*column-name* [, *column-name*] …)]
…REFERENCES *table-name* [(*column-name* [, *column-name*] …)]
…[*action*]
[IN *dbspace-name*]

action:
ON { UPDATE | DELETE { RESTRICT }

location-string:
{ *remote-server-name*.[*db-name*].[*owner*].*object-name*
| *remote-server-name*;[*db-name*];[*owner*];*object-name* }

range-partitioning-scheme:
RANGE(*partition-key*)
(
range-partition-decl [,*range-partition-decl* ...]
)

partition-key:
column-name

range-partition-decl:
partition-name VALUES <= ({*constant-expr* | MAX }) [IN *dbspace-name*]

Examples

Example 1 Creates a table named SalesOrders2 with five columns. Data pages for columns FinancialCode, OrderDate, and ID are in dbspace Dsp3.

Data pages for integer column CustomerID are in dbspace Dsp1. Data pages for CLOB column History are in dbspace Dsp2. Data pages for the primary key, HG for ID, are in dbspace Dsp4.

```
CREATE TABLE SalesOrders2 (
FinancialCode CHAR(2),
CustomerID int IN Dsp1,
History CLOB IN Dsp2,
OrderDate TIMESTAMP,
ID BIGINT
PRIMARY KEY(ID) IN Dsp4
) IN Dsp3
```

Example 2 Creates a table fin_code2 with four columns. Data pages for columns code, type, and id are in the default dbspace, which is determined by the value of the database option DEFAULT_DBSPACE.

Data pages for CLOB column description are in dbspace Dsp2. Data pages from foreign key fk1, HG for c1 are in dbspace Dsp4:

```
CREATE TABLE fin_code2 (
code INT,
type CHAR(10),
description CLOB IN Dsp2,
id BIGINT,
FOREIGN KEY fk1(id) REFERENCES SalesOrders(ID) IN Dsp4
)
```

Example 3 Creates a table t1 where partition p1 is adjacent to p2 and partition p2 is adjacent to p3.

```
CREATE TABLE t1 (c1 INT, c1 INT) (
PARTITION BY RANGE(c1),
(p1 VALUES <= (0), p2 VALUES <= (10), p3 VALUES <= (100))
```

Example 4 Creates a partitioned table bar with six columns and three partitions, mapping data to partitions based on dates.

```
CREATE TABLE bar (
        c1 INT IQ UNIQUE(65500),
        c2 VARCHAR(20),
        c3 CLOB PARTITION (P1 IN Dsp11, P2 IN Dsp12,
           P3 IN Dsp13),
        c4 DATE,
        c5 BIGINT,
        c6 VARCHAR(500) PARTITION (P1 IN Dsp21,
           P2 IN Dsp22),
        PRIMARY KEY (c5) IN Dsp2) IN Dsp1
        PARTITION BY RANGE (c4)
        (P1 VALUES <= ('2006/03/31') IN Dsp31,
         P2 VALUES <= ('2006/06/30') IN Dsp32,
         P3 VALUES <= ('2006/09/30') IN Dsp33
        ) ;
```

Data page allocation for each partition follows:

Partition	Dbspaces	Columns
P1	Dsp31	c1, c2, c4, c5
P1	Dsp11	c3
P1	Dsp21	c6
P2	Dsp32	c1, c2, c4, c5
P2	Dsp12	c3
P2	Dsp22	c6
P3	Dsp33	c1, c2, c4, c5, c6
P3	Dsp13	c3
P1, P2, P3	Dsp1	lookup store of c1 and other shared data
P1, P2, P3	Dsp2	primary key (HG for c5)

Example 5 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)
)
```

Example 6 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 )
)
```

Example 7 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'
```

Example 8 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.

An attempt to create a base table or a global temporary table will fail, if a local temporary table of the same name exists on that connection, as the new table cannot be uniquely identified by *owner.table*.

You can, however, create a local temporary table with the same name as an existing base table or global temporary table. References to the table name access the local temporary table, as local temporary tables are resolved first.

For example, consider the following sequence:

```
CREATE TABLE t1 (c1 int);
INSERT t1 VALUES (9);

DECLARE LOCAL TEMPORARY TABLE t1 (c1 int);
INSERT t1 VALUES (8);

SELECT * FROM t1;
```

The result returned is 8. Any reference to t1 refers to the local temporary table t1 until the local temporary table is dropped by the connection.

You cannot use a temporary table to create a join index.

Do not update a base table that is part of any join index. This is disallowed, and returns the following error:

```
-1000102 Cannot update table %2 because it is defined
in one or more join indexes
```

Sybase IQ does not support the CREATE TABLE ENCRYPTED clause for table level encryption of Sybase IQ tables. The CREATE TABLE ENCRYPTED clause is supported for SQL Anywhere tables in a Sybase IQ database, however.

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.

The IN clauses in *column-definition*, *column-constraint*, *table-constraint*, and *foreign-key* clauses specify the dbspace where the object is to be created. If the IN clause is omitted, Sybase IQ creates the object in the dbspace where the table is assigned.

For more information about dbspaces, see CREATE DBSPACE statement on page 81.

ON COMMIT Allowed for temporary tables only. By default, the rows of a temporary table are deleted on COMMIT.

For clause behavior on multiplex global temporary tables, see “Preserving rows” in Chapter 3, “Running Multiplex Transactions” of *Using Sybase IQ Multiplex*.

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 SQL 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.

In a simplex environment, creating a proxy table which refers to a remote table on the same node is not allowed. In a multiplex environment, creating a proxy table which refers to the remote table defined within the multiplex is not allowed.

For example, in a simplex environment, if you try to create proxy table proxy_e which refers to base table Employees defined on the same node, then the CREATE TABLE ... AT statement is rejected with an error message. In a multiplex environment, the CREATE TABLE .... AT statement is rejected if you create proxy table proxy_e from any node (coordinator or secondary) which refers to remote table Employees defined within a multiplex.

column-definition Defines a column in the table. Allowable data types are described in Chapter 3, "SQL Data Types" in *Reference: Building Blocks, Tables, and Procedures*. 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 *System Administration Guide: Volume 1*.

- **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 Products (
    product_num integer UNIQUE
)
CREATE TABLE Products (
    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,”Chapter 5, “Working with Database Objects,” in the *System Administration Guide: Volume 1*.

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. CHECK condition is *not* enforced by Sybase IQ. For more information about TRUE, FALSE, and UNKNOWN conditions, see “NULL value” and “Search conditions” in Chapter 2, “SQL Language Elements” in *Reference: Building Blocks, Tables, and Procedures*.

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.

PARTITION BY RANGE Specifies that rows are to be partitioned according to the specified ranges of values in the partitioning column.

The *column-name* in the partition-key clause specifies the partition key column. Sybase IQ 15.1 supports a single partition key column.

The *partition-name* in the range-partition-decl clause specifies the name of a new partition on which table rows are stored. Partition names must be unique within the set of partitions on a table. The partition_name clause is required.

VALUE clause Specifies the inclusive upper bound for each partition for range partitioning criteria. The user must specify the partitioning criteria for each range partition to guarantee that each row is distributed to only one partition. NULLs are allowed for the partition column and rows with NULL as partition key value belong to the first table partition. However, NULL cannot be the bound value. There is no lower bound (MIN value) for the first partition. Rows of NULL cells in the first column of the partition key will go to the first partition. For the last partition, you can either specify an inclusive upper bound or MAX. If the upper bound value for the last partition is not MAX, loading or inserting any row with partition key value larger than the upper bound value of the last partition generates an error.

MAX Denotes the infinite upper bound and can only be specified for the last partition.

IN In the *partition-decl*, specifies the dbspace on which rows of the partition should reside.

The following restrictions affect partitions keys and bound values for range partitioned tables:

- Partition bounds must be constants, not constant expressions.
- Partition bounds must be in ascending order according to the order in which the partitions were created. That is, the upper bound for the second partition must be higher than for the first partition, and so on.

  In addition, partition bound values must be compatible with the corresponding partition-key column data type. For example, VARCHAR is compatible with CHAR.
- If a bound value has a different data type than that of its corresponding partition key column, Sybase IQ converts the bound value to the data type of the partition key column, with these exceptions:
- Explicit conversions are not allowed. This example attempts an explicit conversion from INT to VARCHAR and generates an error.

  ```
  CREATE TABLE Employees(emp_name VARCHAR(20))
  PARTITION BY RANGE(emp_name)
  (p1 VALUES <=(CAST (1 AS VARCHAR(20))),
  p2 VALUES <= (CAST (10 AS VARCHAR(20)))
  ```
- Implicit conversions that result in data loss are not allowed. In this example, the partition bounds are not compatible with the partition key type. Rounding assumptions may lead to data loss and an error will be generated.

  ```
  CREATE TABLE emp_id (id INT) PARTITION BY RANGE(id)
  (p1 VALUES <= (10.5), p2 VALUES <= (100.5))
  ```
- In this example, the partition bounds and the partition key data type are compatible. The bound values are directly converted to float values. No rounding is required, and conversion is supported.

  ```
  CREATE TABLE id_emp (id FLOAT)
  PARTITION BY RANGE(id) (p1 VALUES <= (10),
  p2 VALUES <= (100))
  ```

- Conversions from nonbinary datatypes to binary datatypes are not allowed. For example, the following conversion is not allowed and returns an error:

```
CREATE TABLE newemp (name BINARY)
PARTITION BY RANGE(name)
(p1 VALUES <= ("Maarten"),
p2 VALUES <= ("Zymmerman")
```

- NULL cannot be used as a boundary in a range-partitioned table.
- The row will be in the first partition if the cell value of the 1st column of the partition key evaluated to be NULL. Sybase IQ 15.1 supports only single column partition keys, so any NULL in the partition key distributes the row to the first partition.

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 167.
  - **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 349. 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. To create a base table in an IQ main store dbspace, you must have DBA authority or RESOURCE authority and CREATE privilege in the specified dbspace.

See also

ALTER TABLE statement on page 25

Chapter 5, "Working with Database Objects" in *System Administration Guide: Volume 1*

CREATE DBSPACE statement on page 81

CREATE INDEX statement on page 105

"Creating tables" in Chapter 5, "Working with Database Objects," in the *System Administration Guide: Volume 1*

DECLARE LOCAL TEMPORARY TABLE statement on page 167

DROP statement on page 177

"MINIMIZE_STORAGE option" on page 421

CREATE USER statement

Description

Creates a user.

Syntax

CREATE USER *user-name* [**IDENTIFIED BY** *password*]
[**LOGIN POLICY** *policy-name*]
[**FORCE PASSWORD CHANGE** { **ON** | **OFF** }]

Examples

Example 1 The following example creates a user named *SQLTester* with the password welcome. The *SQLTester* user is assigned to the *Test1* login policy and the password expires on the next login.

```
CREATE USER SQLTester IDENTIFIED BY welcome
LOGIN POLICY Test1
FORCE PASSWORD CHANGE ON;
```

Example 2 The following example creates a group named MyGroup

```
CREATE USER MyGroup;
GRANT GROUP TO MyGroup;
```

Usage

user-name The name of the user.

IDENTIFIED BY clause Clause providing the password for the user.

policy-name The name of the login policy to assign the user. No change is made if the LOGIN POLICY clause is not specified.

FORCE PASSWORD CHANGE clause Controls whether the user must specify a new password when they log in. This setting overrides the password_expiry_on_next_login option setting in their policy.

You do not have to specify a password for the user. A user without a password cannot connect to the database. This is useful if you are creating a group and do not want anyone to connect to the database using the group user ID. A user ID must be a valid identifier.

User IDs and passwords cannot:

- Begin with white space, single quotes, or double quotes
- End with white space
- Contain semicolons

A password can be either a valid identifier, or a string (maximum 255 bytes) placed in single quotes. Passwords are case sensitive. It is recommended that the password be composed of 7-bit ASCII characters, as other characters may not work correctly if the database server cannot convert them from the client's character set to UTF-8.

The VERIFY_PASSWORD_FUNCTION option can be used to specify a function to implement password rules (for example, passwords must include at least one digit). If a password verification function is used, you cannot specify more than one user ID and password in the GRANT CONNECT statement. For details, see "VERIFY_PASSWORD_FUNCTION option" on page 469 and "GRANT statement" on page 206.

Side Effects None.

Standards

- **SQL2003** Vendor extension.
- **Sybase** Not supported by Adaptive Server Enterprise.

Permissions

Must be owner of the view or have DBA authority.

See also

"ALTER LOGIN POLICY statement" on page 19

"CREATE USER statement" on page 151

"COMMENT statement" on page 61

"CREATE LOGIN POLICY statement" on page 117

"DROP LOGIN POLICY statement" on page 183

"DROP USER statement" on page 185

"GRANT statement" on page 206.

"Managing login policies overview" in *SQL Anywhere Server – Database Administration > Configuring Your Database > Managing user IDs, authorities, and permissions*

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 1, "Using Procedures and Batches" in the *System Administration Guide: Volume 2.*

Side effects

None.

Standards

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

Permissions

None.

See also

BEGIN … END statement on page 47

Chapter 3, "SQL Data Types" in *Reference: Building Blocks, Tables, and Procedures*

DECLARE statement on page 158

DROP VARIABLE statement on page 186

SET statement [ESQL] on page 301

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

Example 1 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 Employees
WHERE Sex = 'M'
```

Example 2 Creates a view showing employees and the departments they belong to:

```
CREATE VIEW emp_dept
AS SELECT Surname, GivenName, DepartmentName
FROM Employees JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID
```

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 291). 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 291).

AS The SELECT statement on which the view is based must not contain an ORDER BY clause, a subquery in the SELECT list, or a TOP or FIRST qualification. 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 135

DROP statement on page 177

"Copy Definition utility (defncopy)" in Chapter 3, "Database Administration Utilities" in the *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 4.

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 307

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 “Embedded SQL programming techniques” in *SQL Anywhere Server – Programming > SQL Anywhere Data Access APIs > SQL Anywhere embedded SQL* for more information.

Standards
- **SQL92**
- **Sybase**

Permissions
None.

See also
BEGIN … END statement on page 47

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 varname;
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

Example 1 Illustrates how to declare a scroll cursor in Embedded SQL:

```
EXEC SQL DECLARE cur_employee SCROLL CURSOR
FOR SELECT * FROM Employees;
```

Example 2 Illustrates how to declare a cursor for a prepared statement in Embedded SQL:

```
EXEC SQL PREPARE employee_statement
FROM 'SELECT emp_lname FROM Employees';
EXEC SQL DECLARE cur_employee CURSOR
FOR employee_statement ;
```

Example 3 Illustrates the use of cursors in a stored procedure:

```
BEGIN
  DECLARE cur_employee CURSOR FOR
      SELECT emp_lname
      FROM Employees;
  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.

A cursor declared FOR READ ONLY sees the version of table(s) on which the cursor is declared when the cursor is opened, not the version of table(s) at the time of the first FETCH.

For example,

```
CREATE TABLE t1 ( c1 INT );
INSERT t1 VALUES ( 1 );

BEGIN
DECLARE t1_cursor CURSOR FOR SELECT * FROM t1
FOR READ ONLY;
OPEN t1_cursor;
INSERT t1 VALUES ( 2 );
FETCH T1_CURSOR;
END
```

When the cursor is fetched, only one row can be fetched from the table.

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.ISYSTAB '
||
            '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 SQL Anywhere support of updatable cursors. For a full discussion of cursor types and working with cursors, see "Introduction to cursors" in *SQL Anywhere Server – Programming > Introduction to Programming with SQL Anywhere > Using SQL in applications*. 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 trade-off between efficiency and consistency. Asensitive cursors provide efficient performance at the expense of consistency.

Sybase IQ supports updatable cursors on single tables.

Scalar user-defined functions and user-defined aggregate functions are not supported in updatable cursors.

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
- Operator that has an OLAP function, with the exception of RANK()

See the description of the UPDATE (positioned) statement [ESQL] [SP] on page 326 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)
```

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 55

DELETE (positioned) statement [ESQL] [SP] on page 171

OPEN statement [ESQL] [SP] on page 260

PREPARE statement [ESQL] on page 268

SELECT statement on page 291

UPDATE (positioned) statement [ESQL] [SP] on page 326

sp_iqcursorinfo procedure in Chapter 7, “System Procedures” in *Reference: Building Blocks, Tables, and Procedures*

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 159.

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 159

sp_iqcursorinfo procedure in Chapter 7, “System Procedures” in *Reference: Building Blocks, Tables, and Procedures*

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

Example 1 Illustrates how to declare a local temporary table in Embedded SQL:

```
EXEC SQL DECLARE LOCAL TEMPORARY TABLE MyTable (
  number INT
   );
```

Example 2 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 135 for definitions of *column-definition*, *column-constraint*, and *table-constraint*, and the NOT TRANSACTIONAL clause. See SELECT statement on page 291 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 can create a local temporary table implicitly by entering:

```
select * into #tmp from table1
```

Or you can 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);
```

An attempt to create a base table or a global temporary table will fail, if a local temporary table of the same name exists on that connection, as the new table cannot be uniquely identified by *owner.table*.

You can, however, create a local temporary table with the same name as an existing base table or global temporary table. References to the table name access the local temporary table, as local temporary tables are resolved first.

For example, consider the following sequence:

```
CREATE TABLE t1 (c1 int);
INSERT t1 VALUES (9);

DECLARE LOCAL TEMPORARY TABLE t1 (c1 int);
INSERT t1 VALUES (8);

SELECT * FROM t1;
```

The result returned is 8. Any reference to t1 refers to the local temporary table t1 until the local temporary table is dropped by the connection.

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 135

SELECT statement on page 291

DELETE statement

Description

Deletes rows from the database.

Syntax

DELETE [**FROM**] [*owner*.]*table-name*
…[**FROM** *table-list*]
…[**WHERE** *search-condition*]

Examples

Example 1 Removes employee 105 from the database:

```
DELETE
FROM Employees
WHERE EmployeeID = 105
```

Example 2 Removes all data prior to 1993 from the FinancialData table:

```
DELETE
FROM FinancialData
WHERE Year < 1993
```

Example 3 Removes all names from the Contacts table if they are already present in the Customers table:

```
DELETE
FROM Contacts
FROM Contacts, Customers
WHERE Contacts.Surname = Customers.Surname
AND Contacts.GivenName = Customers.GivenName
```

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 200

INSERT statement on page 216

SYNCHRONIZE JOIN INDEX statement on page 318

TRUNCATE TABLE statement on page 319

DELETE (positioned) statement [ESQL] [SP]

Description

Deletes the data at the current location of a cursor.

Syntax

DELETE [**FROM** *table-spec*]
WHERE CURRENT OF *cursor-name*

Parameters

cursor-name:
identifier | *hostvar*

table-spec:
[*owner*.]*correlation-name*

owner:
identifier

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 table from which rows are deleted is determined as follows:

- If no FROM clause is included, the cursor must be on a single table only.
- If the cursor is for a joined query (including using a view containing a join), then the FROM clause must be used. Only the current row of the specified table is deleted. The other tables involved in the join are not affected.
- If a FROM clause is included, and no table owner is specified, *table-spec* is first matched against any correlation names.
    - If a correlation name exists, *table-spec* is identified with the correlation name.
    - If a correlation name does not exist, *table-spec* must be unambiguously identifiable as a table name in the cursor.
- If a FROM clause is included, and a table owner is specified, table-spec must be unambiguously identifiable as a table name in 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 SQL Anywhere.

Permissions

Must have DELETE permission on tables used in the cursor.

See also

DECLARE CURSOR statement [ESQL] [SP] on page 159

INSERT statement on page 216

UPDATE statement on page 322

UPDATE (positioned) statement [ESQL] [SP] on page 326

sp_iqcursorinfo procedure in Chapter 7, “System Procedures” in *Reference: Building Blocks, Tables, and Procedures*

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 ALLOCATE DESCRIPTOR statement [ESQL] on page 4).

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, or on a cursor that is not a dynamic cursor, 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.

INPUT 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 "Embedded SQL programming techniques" in *SQL Anywhere Server – Programming > SQL Anywhere Data Access APIs > SQL Anywhere embedded SQL*.

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 159

OPEN statement [ESQL] [SP] on page 260

PREPARE statement [ESQL] on page 268

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

Example 1 The following statement shows how to use DISCONNECT in Embedded SQL:

```
EXEC SQL DISCONNECT :conn_name
```

Example 2 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 65

SET CONNECTION statement [DBISQL] [ESQL] on page 306

DROP statement

Description

Removes objects from the database.

Syntax

DROP
{ **DBSPACE** *dbspace-name*
| { **DATATYPE** [**IF EXISTS**]
| **DOMAIN** [**IF EXISTS**] } *datatype-name*
| **EVENT** [**IF EXISTS**] *event-name*
| **INDEX** [**IF EXISTS**] [[*owner*].*table-name*.]*index-name*
| **JOIN INDEX** [*owner*.]*join-index-name*
| **MESSAGE** *message-number*
| **TABLE** [**IF EXISTS**] [*owner*.]*table-name*
| **VIEW** [**IF EXISTS**] [*owner*.]*view-name*
| **PROCEDURE** [**IF EXISTS**] [*owner*.]*procedure-name*
| **FUNCTION** [**IF EXISTS**] [*owner*.]*function-name* }

Examples

Example 1 Drops the Departments table from the database:

```
DROP TABLE Departments
```

Example 2 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.

Use the IF EXISTS clause if you do not want an error returned when the DROP statement attempts to remove a database object that does not exist.

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. You cannot drop these initial dbspaces, but you may drop dbspaces from the IQ main store or catalog store, which may contain multiple dbspaces, as long as at least one dbspace remains with readwrite mode.

You must 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. You can drop a dbspace only after you make the dbspace read-only.

Note A dbspace may contain data at any point after it is used by a command, thereby preventing a DROP DBSPACE on it.

For more information on modifying dbspaces, see “Working with dbspaces” in Chapter 5, “Working with Database Objects” in the *System Administration Guide: Volume 1*.

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.

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 9

ALTER TABLE statement on page 25

CREATE DBSPACE statement on page 81

CREATE DOMAIN statement on page 84

CREATE EVENT statement on page 86

CREATE INDEX statement on page 105

CREATE MESSAGE statement [T-SQL] on page 119

CREATE PROCEDURE statement on page 120

CREATE TABLE statement on page 135

CREATE VIEW statement on page 155

sp_iqdbspace procedure in Chapter 7, "System Procedures" in *Reference: Building Blocks, Tables, and Procedures*

Chapter 5, "Working with Database Objects" in the *System Administration Guide: Volume 1*

DROP CONNECTION statement

Description

Drops connection of any user 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. You cannot drop your current connection; you must first create another connection, then drop your first connection.

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 65

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
Example 1 Drops database mydb:

```
DROP DATABASE 'mydb.db'
```

Example 2 Drops the encrypted database *marvin.db*, which was created with the key *is!seCret*:

```
DROP DATABASE 'marvin.db' KEY 'is!seCret'
```

Example 3 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.

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 68

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

Must have DBA authority.

See also

CREATE EXTERNLOGIN statement on page 96

DROP LOGIN POLICY statement

Description

Removes a login policy from the database.

Syntax

DROP LOGIN POLICY *policy-name*

Examples

The following example creates the *Test11* login policy and then deletes it.

```
CREATE LOGIN POLICY Test11;
DROP LOGIN POLICY Test11 ;
```

Usage

A DROP LOGIN POLICY statement fails if you attempt to drop a policy that is assigned to a user. You can use either the ALTER USER statement to change the user's policy assignment or DROP USER to drop the user.

Permissions

Must have DBA authority.

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 will succeed.

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 130

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 23

CREATE SERVICE statement on page 132

“Introduction to web services” in *SQL Anywhere Server – Programming > SQL Anywhere Data Access APIs > SQL Anywhere web services*

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 268

DROP USER statement

Description

Removes a user.

Syntax

DROP USER *user-name*

Examples

The following example drops a user named *SQLTester* from the database.

```
DROP USER SQLTester;
```

Usage

user-name The name of the user.

Side effects

None.

Standards

- **SQL2003** Vendor extension.
- **Sybase** Not supported by Adaptive Server Enterprise.

Permissions

Must have DBA authority.

See also

"ALTER LOGIN POLICY statement" on page 19

"CREATE USER statement" on page 151

"CREATE LOGIN POLICY statement" on page 117

"DROP LOGIN POLICY statement" on page 183

"GRANT statement" on page 206

"Managing login policies overview" in *SQL Anywhere Server – Database Administration > Configuring Your Database > Managing user IDs, authorities, and permissions*

DROP VARIABLE statement

Description

Eliminates a SQL variable.

Syntax

DROP VARIABLE *identifier*

Usage

The DROP VARIABLE statement eliminates a SQL variable that was previously created using the CREATE VARIABLE statement. Variables are automatically eliminated when the database connection is released. Variables are often used for large objects, so eliminating them after use or setting them to NULL can free up significant resources (primarily disk space).

Use the IF EXISTS clause if you do not want an error returned when the DROP statement attempts to remove a database object that does not exist.

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not supported in Adaptive Server Enterprise.

Permissions

None

See also

CREATE VARIABLE statement on page 153

SET statement [ESQL] on page 301

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

Example 1 Executes a DELETE:

```
EXEC SQL EXECUTE IMMEDIATE
'DELETE FROM Employees WHERE EmployeeID = 105';
```

Example 2 Executes a prepared DELETE statement:

```
EXEC SQL PREPARE del_stmt FROM
'DELETE FROM Employees WHERE EmployeeID = :a';
EXEC SQL EXECUTE del_stmt USING :employee_number;
```

Example 3 Executes a prepared query:

```
EXEC SQL PREPARE sel1 FROM
'SELECT Surname FROM Employees WHERE EmployeeID = :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 and is dropped on completion.

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 159

PREPARE statement [ESQL] on page 268

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 @intvar 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 55

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 441

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 47

CREATE PROCEDURE statement on page 120

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. The default action is to COMMIT any changes you have made to the database.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not applicable in Adaptive Server Enterprise.

Permissions

None

See also

SET OPTION statement on page 307

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

Example 1 An Embedded SQL example:

```
EXEC SQL DECLARE cur_employee CURSOR FOR
SELECT EmployeeID, Surname FROM Employees;
EXEC SQL OPEN cur_employee;
EXEC SQL FETCH cur_employee
INTO :emp_number, :emp_name:indicator;
```

Example 2 A procedure example:

```
BEGIN
   DECLARE cur_employee CURSOR FOR
      SELECT Surname
      FROM Employees;
   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 1, “Using Procedures and Batches” in the *System Administration Guide: Volume 2*.

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 handles the FIRST, LAST, ABSOLUTE, and negative RELATIVE options less efficiently than some other DBMS products, so there is a performance impact when using them.

OPEN initially positions the cursor before the first row.

A cursor declared FOR READ ONLY sees the version of table(s) on which the cursor is declared when the cursor is opened, not the version of table(s) at the time of the first FETCH

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 and OPEN statements 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 370

DECLARE CURSOR statement [ESQL] [SP] on page 159

OPEN statement [ESQL] [SP] on page 260

PREPARE statement [ESQL] on page 268

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-type*] **CURSOR**
{ **FOR** *statement*
... [{ **FOR** { **UPDATE** *cursor-concurrency* | **FOR READ ONLY** }]
| **USING** *variable-name* }
DO *statement-list*
END FOR [*statement-label*]

Parameters

cursor-type:
NO SCROLL
| DYNAMIC SCROLL
| SCROLL
| INSENSITIVE
| SENSITIVE

cursor-concurrency:
BY { VALUES | TIMESTAMP | LOCK]

variable-name:
identifier

Examples

The following fragment illustrates the use of the FOR loop:

```
FOR names AS curs CURSOR FOR
SELECT Surname
FROM Employees
DO
    CALL search_for_name( Surname );
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.

For descriptions of the *cursor-type* parameters and more examples, see "FOR statement" in *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (E-O)*.

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 159

FETCH statement [ESQL] [SP] on page 193

LEAVE statement on page 229

LOOP statement on page 255

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 native syntax of the remote server. The command or group of commands is enclosed in curly braces ({}) or single quotes.

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.

Note The FORWARD TO statement is a server directive and cannot be used in stored procedures, triggers, events, or batches.

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 130

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

Example 1 The following are valid FROM clauses:

```
...
FROM Employees
...
...
FROM Employees NATURAL JOIN Departments
...
...
FROM Customers
KEY JOIN SalesOrders
KEY JOIN SalesOrderItems
KEY JOIN Products
...
```

Example 2 The following query illustrates how to use derived tables in a query:

```
SELECT Surname, GivenName, number_of_orders
FROM Customers JOIN
     ( SELECT CustomerID, count(*)
       FROM SalesOrders
        GROUP BY CustomerID )
     AS sales_order_counts ( CustomerID,
                             number_of_orders )
ON ( Customers.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.

A SELECT statement can also return a result set from a procedure. Note that CIS functional compensation performance considerations apply. For syntax and an example, see “FROM clause” in *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (E-O)*.

The *join-type* keywords are described in Table 1-9.

Table 1-9: 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 Products pr, SalesOrders so, SalesOrderItems si
  WHERE pr.ProductID = so.ProductID
    AND pr.ProductID = si.ProductID;
```

The same query can use the preferable keyword-style join:

```
SELECT *
  FROM Products pr INNER JOIN SalesOrders so
    ON (pr.ProductID = so.ProductID)
  INNER JOIN SalesOrderItems si
    ON (pr.ProductID = si.ProductID);
```

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” in *Reference: Building Blocks, Tables, and Procedures*.

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 SalesOrders
KEY JOIN SalesOrderItems,
SalesOrders
KEY JOIN Employees
```

The two instances of the SalesOrders table are treated as one instance that is equivalent to:

```
SELECT *
FROM SalesOrderItems
KEY JOIN SalesOrders
KEY JOIN Employees
```

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 SQL 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 SQL Anywhere documentation for rules that might apply to processing.

If you have a query that does not require a FROM clause, you can force 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 169

SELECT statement on page 291

"Search conditions" in Chapter 2, "SQL Language Elements" in *Reference: Building Blocks, Tables, and Procedures*

Chapter 2, "Using OLAP," in the *System Administration Guide: Volume 2*

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 4.

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. LONG VARCHAR and LONG BINARY are not supported by GET DESCRIPTOR ... DATA.

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 156

SET DESCRIPTOR statement [ESQL] on page 307

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**] [**FROM** *userid*]

Syntax 4

GRANT EXECUTE ON [*owner*.]*procedure-name* **TO** *userid* [, …]

Syntax 5

GRANT INTEGRATED LOGIN TO *user_profile_name* [, …] **AS USER** *userid*

Syntax 6

GRANT CREATE ON *dbspace_name* **TO** *userid* [, …]

Syntax 7

GRANT KERBEROS LOGIN TO *client-Kerberos-principal, ...*

AS USER *userid*

Examples

Example 1 Makes two new users for the database:

```
GRANT
CONNECT TO Laurel, Hardy
IDENTIFIED BY Stan, Ollie
```

Example 2 Grants permissions on the Employees table to user Laurel:

```
GRANT
SELECT, INSERT, DELETE
ON Employees
TO Laurel
```

Example 3 Allows the user Hardy to execute the Calculate_Report procedure:

```
GRANT
EXECUTE ON Calculate_Report
TO Hardy
```

Example 4 Gives users Lawrence and Swift CREATE permission on dbspace *DspHist*:

```
GRANT
CREATE ON DspHist
TO LAWRENCE, SWIFT
```

Example 5 Grants CREATE privilege on dbspace DspHist to users Fiona and Ciaran:

```
GRANT CREATE ON DspHist TO Fiona, Ciaran
```

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:

Syntax 6 gives CREATE permission on the specified dbspace to the specified user(s) and/or group(s).

CONNECT TO userid,... Creates a new user. GRANT CONNECT can also be used by any user to change their own password.

Note Sybase recommends using the CREATE USER statement to create users. See CREATE USER statement on page 151.

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.

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" in Chapter 2, "SQL Language Elements" in *Reference: Building Blocks, Tables, and Procedures*. 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 *userid* and *password* with the GRANT CONNECT statement. For details, see "VERIFY_PASSWORD_FUNCTION option" on page 469.

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 8, "Managing User IDs and Permissions," in the *System Administration Guide: Volume 1* 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.

If you do not want a specific user to access a particular table, view, or procedure, then do not make that user a member of a group that has permissions on that object.

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 Employees table to user Laurel, enter:

```
GRANT
SELECT, UPDATE ( street )
ON Employees
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.

Syntax 6 grants CREATE permission to the specified user(s) or group(s).

Syntax 7 creates a Kerberos authenticated login mapping from one or more Kerberos principals to an existing database user ID. This allows users who have successfully logged in to Kerberos (users who have a valid Kerberos ticket-granting ticket) to connect to a database without having to provide a user ID or password. For more information on this syntax, see "GRANT statement" in *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (E-O)*.

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.

- For Syntax 6, you must have DBA authority.
- For Syntax 7, you must have DBA authority.

See also

REVOKE statement on page 287

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 CompanyName, CAST(
sum(SalesOrderItems.Quantity *
    Products.UnitPrice) AS INTEGER) VALUE
    FROM Customers
    LEFT OUTER JOIN SalesOrders
    LEFT OUTER JOIN SalesOrsderItems
    LEFT OUTER JOIN Product
    GROUP BY CompanyName ;


    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” in Chapter 2, “SQL Language Elements” in *Reference: Building Blocks, Tables, and Procedures*.

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 47

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

Example 1 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
```

Example 2 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).

Syntax

Syntax 1

INSERT [**INTO**] [*owner*.]*table-name* [(*column-name* [, …])]
... { **DEFAULT VALUES** | **VALUES** ([*expression* | **DEFAULT**,…)] }

Syntax 2

INSERT [**INTO**] [*owner*.]*table-name* [(*column-name* [, …])]
... *insert-load-options insert-select-load-options*
... *select-statement*

Syntax 3

INSERT [**INTO**] [*owner*.]*table-name*[(*column-name* [, …])]
... *insert-load-options insert-select-load-options*
LOCATION '*servername.dbname*'
[*location-options*]
... { { *select-statement* } | '*select statement*' }

Parameters

insert-load-options:
[LIMIT *number-of-rows*]
[NOTIFY *number-of-rows*]
[SKIP *number-of-rows*]
[START ROW ID *number*]

insert-select-load-options:
[WORD SKIP *number*]
[IGNORE CONSTRAINT *constrainttype* [, …]]
[MESSAGE LOG '*string*' ROW LOG '*string*' [ONLY LOG *logwhat* [, …]]]
[LOG DELIMITED BY '*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 }

location-options:
[ENCRYPTED PASSWORD]
[PACKETSIZE *packet-size*]
[QUOTED_IDENTIFIER { ON | OFF }]
[ISOLATION LEVEL { READ UNCOMMITTED | READ
COMMITTTED | SERIALIZABLE }]

Examples

Example 1 Adds an Eastern Sales department to the database:

```
INSERT INTO Departments
(DepartmentID, DepartmentName, DepartmentHeadID)
VALUES (600, 'Eastern Sales', 501)
```

Example 2 Fills the table dept_head with the names of department heads and their departments:

```
INSERT INTO dept_head (name, dept)
   NOTIFY 20
   SELECT Surname || ' ' || GivenName
   AS name,
   dept_name
FROM Employees JOIN Departments
   ON EmployeeID= DepartmentHeadID
```

Example 3 Inserts data from the l_shipdate and l_orderkey columns of the lineitem table from the Sybase IQ database iqdet on the remote server detroit into the corresponding columns of the lineitem table in the current database:

```
INSERT INTO lineitem
   (l_shipdate, l_orderkey)
   LOCATION 'detroit.iqdet'
   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 4, "SQL Functions" in *Reference: Building Blocks, Tables, and Procedures*.

Syntax 3 INSERT...LOCATION is a variation of Syntax 2 that allows you to insert data from an Adaptive Server Enterprise or Sybase IQ database. The *servername.dbname* specified in the LOCATION clause identifies the remote server and database for the table in the FROM clause. To use Syntax 3, the Adaptive Server Enterprise or Sybase IQ remote server to which you are connecting must exist in the Sybase Open Client *interfaces* or *sql.ini* file on the local machine.

In queries using Syntax 3, you can insert a maximum of 2147483647 rows.

The SELECT statement can 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.)

The local Sybase IQ server connects to the server and database you specify in the LOCATION clause. The results from the queries on the remote tables are returned and the local server inserts the results in the current database. If you do not specify a server name in the LOCATION clause, Sybase IQ ignores any database name you 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.

Creating a remote login with the CREATE EXTERNLOGIN statement and defining a remote server with a CREATE SERVER statement sets up an external login and password for INSERT...LOCATION such that any user can use the login and password in any context. This avoids possible errors due to inaccessibility of the login or password.

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.

Note Password encryption requires Open Client 15.0.

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.

When INSERT...LOCATION is transferring data between a Sybase IQ server and a remote Sybase IQ or Adaptive Server Enterprise server, the value of the INSERT...LOCATION TDS PACKETSIZE parameter is always equal to 512 bytes, even if you specify a different value for PACKETSIZE.

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};
```

The ISOLATION LEVEL parameter allows you to specify an isolation level for the connection to a remote server.

Isolation level	Characteristics
READ UNCOMMITTED	• Isolation level 0 • Read permitted on row with or without write lock • No read locks are applied • No guarantee that concurrent transaction will not modify row or roll back changes to row
READ COMMITTED	• Isolation level 1 • Read only permitted on row with no write lock • Read lock acquired and held for read on current row only, but released when cursor moves off the row • No guarantee that data will not change during transaction

Isolation level	Characteristics
SERIALIZABLE	• Isolation level 3 • Read only permitted on rows in result without write lock • Read locks acquired when cursor is opened and held until transaction ends

For more information on isolation levels, see "Isolation levels and consistency" in *SQL Anywhere Server – SQL Usage > Creating Databases > Using transactions and isolation levels*.

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. By default, if a remote data column contains over 2GB, Sybase IQ silently truncates the column value to 2GB.

Warning! Sybase IQ does not support the Adaptive Server Enterprise data types UNICHAR, UNIVARCHAR, or UNITEXT. If an INSERT...LOCATION command from UNICHAR or UNITEXT to CHAR or CLOB columns in the ISO_BINENG collation executes without error, the data in the columns may be inconsistent. An error is reported in this situation, only if the conversion fails.

Users must be specifically licensed to use the Large Objects Management functionality. For details on the Large Objects Management option, see *Large Objects Management in Sybase IQ*.

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 *System Administration Guide: Volume 1*.

The DEFAULT VALUES and VALUES clauses allow you to specify the values to insert. If you want to insert the default column values as specified in the CREATE TABLE statement, specify DEFAULT VALUES. Specifying DEFAULT VALUES is semantically equivalent to specifying the following explicit syntax:

```
INSERT [INTO} <tablename>
VALUES(default, default, ..., default)
```

where the number of default entries is equal to the number of columns in the table.

You can also use the INSERT VALUES(DEFAULT ...) clause to insert into NULL columns.

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 *System Administration Guide: Volume 1*.

The START ROW ID clause of the LOAD TABLE and the INSERT commands is not allowed on a partitioned table.

Note Use the START ROW ID option for partial-width inserts only. If the columns being loaded already contain data, the insert fails.

For information on the *insert-select-load-options* WORD SKIP, IGNORE CONSTRAINT, MESSAGE LOG, ROW LOG, and LOG DELIMITED BY and the *constrainttype* and *logwhat* parameters, see the LOAD TABLE statement on page 230.

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 *System Administration Guide: Volume 1*.

An INSERT from a stored procedure or function is not permitted, if the procedure or function uses COMMIT, ROLLBACK, or some ROLLBACK TO SAVEPOINT statements. For more information, see “Atomic compound statements” and “Transactions and savepoints in procedures” in Chapter 1, “Using Procedures and Batches” of the *System Administration Guide: Volume 2*.

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 169

LOAD TABLE statement on page 230

SYNCHRONIZE JOIN INDEX statement on page 318

“Using the INSERT statement” in Chapter 7, “Moving Data In and Out of Databases” in the *System Administration Guide: Volume 1*

INSTALL JAVA 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

Example 1 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
```

Example 2 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).

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 JAR is retained as a JAR after the classes that it contains have been installed. That JAR is the associated JAR of 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 277

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

“sp_iqsysmon procedure” in Chapter 7, “System Procedures” in *Reference: Building Blocks, Tables, and Procedures*

MONITOR_OUTPUT_DIRECTORY option on page 422

Chapter 5, “Monitoring and Tuning Performance” in the *Performance and Tuning Guide* for examples of monitor results

Chapter 1, “Using Procedures and Batches” in *System Administration Guide: Volume 2* 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

Example 1 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
```

Example 2 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 47

FOR statement on page 197

LOOP statement on page 255

LOAD TABLE statement

Description Imports data into a database table from an external file.

Syntax

LOAD [**INTO**] **TABLE** [*owner*.]*table-name*
... (*load-specification* [, …])
... { **FROM** | **USING** [**CLIENT**] **FILE** }
{ '*filename-string*' | *filename-variable* } [, …]
... [**CHECK CONSTRAINTS** { **ON** | **OFF** }]
... [**DEFAULTS** { **ON** | **OFF** }]
... [**QUOTES OFF**]
... **ESCAPES OFF**
... [**FORMAT** { **ascii** | **binary** | **bcp** }]
... [**DELIMITED BY** '*string*']
... [**STRIP** { **ON** | **OFF** | **RTRIM** }]
... [**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**]
... [**ON PARTIAL INPUT ROW** { **ROLLBACK** | **CONTINUE** }]
... [**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*)
| ENCRYPTED (*data-type* '*key-string*' [, '*algorithm-string*'])
| DEFAULT *default-value* }
[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

Example 1 Loads data from one file into the Products table on a Windows system. A tab is used as the column delimiter following the Description and Color columns.

```
LOAD TABLE Producis
( IDASCII(6),
FILLER(1),
Name   ASCII(15),
FILLER(1),
Description   '\x09',
Size   ASCII(2),
FILLER(1),
Color   '\x09',
Quantity   PREFIX 2,
UnitPrice   PREFIX 2,
FILLER(2) )
FROM 'C:\\mydata\\source1.dmp'
QUOTES OFF
ESCAPES OFF
BYTE ORDER LOW
NOTIFY 1000
```

Example 2 Loads data from a file, *a.inp*, on a client computer.

```
LOAD TABLE t1(c1,c2,filler(30))
USING CLIENT FILE 'c:\\client-data\\a.inp'
QUOTES OFF ESCAPES OFF
IGNORE CONSTRAINT UNIQUE 0, NULL 0
MESSAGE LOG 'c:\\client-data\\m.log'
ROW LOG 'c:\\client-data\\r.log'

ONLY LOG UNIQUE
```

Example 3 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'
```

Example 4 Ignores 10 word-length violations; on the 11th, deploys 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
```

Example 5 Loads data into table t1from the BCP character file *bcp_file.bcp* using the FORMAT BCP load option:

```
LOAD TABLE t1 (c1, c2, c3)
FROM ‘bcp_file.bcp’
FORMAT BCP
...
```

Example 6 The following LOAD TABLE statement loads default values 12345 into c1 using the DEFAULT load option and loads c2 and c3 with data from the *LoadConst04.dat* file:

```
LOAD TABLE t1 (c1 DEFAULT ‘12345 ’, c2, c3, filler(1))
FROM ‘LoadConst04.dat’
STRIP OFF
QUOTES OFF
ESCAPES OFF
DELIMITED BY ‘,’;
```

Example 7 The following LOAD TABLE statement loads c1 and c2 with data from the file *bcp_file.bcp* using the FORMAT BCP load option and sets c3 to the value 10.

```
LOAD TABLE t1 (c1, c2, c3 DEFAULT '10')
FROM 'bcp_file.bcp'
FORMAT BCP
QUOTES OFF
ESCAPES 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.

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. Sybase IQ does not 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 processed in the order 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 can 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. For BCP, the delimiter can be up to 10 characters.

- 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 1-10: 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".)

Option	Meaning
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 1-11: 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 *System Administration Guide: Volume 1*.

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 in Chapter 3, “SQL Data Types” in *Reference: Building Blocks, Tables, and Procedures* or Chapter 7, “Moving Data In and Out of Databases” in the *System Administration Guide: Volume 1*.

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 TABLE to insert a NULL value in col1 when the data is 000000, either write the NULL clause as `null('000000')`, or modify the data to equal binary zeros and use NULL(ZEROS).

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.

Use the DEFAULT option to specify a load default column value. You can load a default value into a column, even if the column does not have a default value defined in the table schema. This feature provides more flexibility at load time.

- The LOAD TABLE DEFAULTS option must be ON in order to use the default value specified in the LOAD TABLE statement. If the DEFAULTS option is OFF, the specified load default value is not used and a NULL value is inserted into the column instead.
- The LOAD TABLE command must contain at least one column that needs to be loaded from the file specified in the LOAD TABLE command. Otherwise, an error is reported and the load is not performed.
- The specified load default value must conform to the supported default values for columns and default value restrictions as described in the section “Using column defaults,” in Chapter 9, “Ensuring Data Integrity,” of the *System Administration Guide: Volume 1*. The LOAD TABLE DEFAULT option does not support AUTOINCREMENT, IDENTITY, or GLOBAL AUTOINCREMENT as a load default value.
- The LOAD TABLE DEFAULT *default-value* must be of the same character set as that of the database.
- Encryption of the default value is not supported for the load default values specified in the LOAD TABLE DEFAULT clause.
- A constraint violation caused by evaluation of the specified load default value is counted for each row that is inserted in the table.

Another important part of the *load-specification* is the FILLER option. This option 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 Employees table is:

  ```
  LOAD TABLE Employees
  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.

Descriptions of each statement clause follow:

USING USING FILE loads one or more files from the server. This clause is synonymous with specifying the FROM *filename* clause. USING CLIENT FILE bulk loads one or more files from a client. The character set of the file on the client side must be the same as the server collation. Sybase IQ processes files in the file list serially. Each file is locked in read mode as it is processed, then unlocked. Client-side bulk loading incurs no administrative overhead such as extra disk space, memory or network-monitoring daemon requirements.

When bulk loading large objects, the USING CLIENT FILE clause applies to both primary and secondary files. (If you have the Large Objects Management Option, see *Large Objects Management in Sybase IQ* for details.)

During client-side loads, the IGNORE CONSTRAINT log files are created on the client host and any error while creating the log files causes the operation to roll back.

Client-side bulk loading is supported by Interactive SQL and ODBC/JDBC clients using the Command Sequence protocol. It is not supported by clients using the TDS protocol. For data security over a network, use Transport Layer Security. To control who can use client-side bulk loads, use the secure feature (-sf) server startup switch, the ALLOW_READ_CLIENT_FILE database option, and/or the READCLIENTFILE access control. For details, see *SQL Anywhere Server – SQL Usage*.

The LOAD TABLE FROM clause is deprecated, but may be used to specify a file that exists on the server.

The following example loads data from the file *a.inp* on a client computer.

```
LOAD TABLE t1(c1,c2,filler(30))
USING CLIENT FILE 'c:\\client-data\\a.inp'
QUOTES OFF ESCAPES OFF
IGNORE CONSTRAINT UNIQUE 0, NULL 0
MESSAGE LOG 'c:\\client-data\\m.log'
ROW LOG 'c:\\client-data\\r.log'
ONLY LOG UNIQUE
```

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 *System Administration Guide: Volume 1*.

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 Street, 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. (See “STRIP” on page 243 for more information.) 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 extract data to be loaded into an IQ table and the string fields 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 *System Administration Guide: Volume 1*.

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.

For a detailed description of the binary format used by Sybase IQ to produce data files that can be read by the LOAD TABLE statement using the FORMAT BINARY and BINARY column specification clauses, see Sybase IQ binary load format in Chapter 3, "SQL Data Types" of *Reference: Building Blocks, Tables, and Procedures*.

Sybase IQ also accepts data from BCP character files as input to the LOAD TABLE command.

- The BCP data file loaded into Sybase IQ tables using the LOAD TABLE FORMAT BCP statement must be exported (BCP OUT) in cross-platform file format using the -c option.
- For FORMAT BCP, the default column delimiter for the LOAD TABLE statement is <tab> and the default row terminator is <newline>.
- For FORMAT BCP, the last column in a row must be terminated by the row terminator, not by the column delimiter. If the column delimiter is present before the row terminator, then the column delimiter is treated as a part of the data.
- Data for columns that are not the last column in the load specification must be delimited by the column delimiter only. If a row terminator is encountered before a column delimiter for a column that is not the last column, then the row terminator is treated as a part of the column data.
- Column delimiter can be specified via the DELIMITED BY clause. For FORMAT BCP, the delimiter must be less than or equal to 10 characters in length. An error is returned, if the delimiter length is more than 10.
- For FORMAT BCP, the load specification may contain only column names, NULL, and ENCRYPTED. An error is returned, if any other option is specified in the load specification.

  For example, the following LOAD TABLE load specifications are valid:

  ```
  LOAD TABLE x( c1, c2 null(blanks), c3 )
  FROM 'bcp_file.bcp'
  FORMAT BCP
  ...

  LOAD TABLE x( c1 encrypted(bigint,'KEY-ONE','aes'),
  c2, c3 )
  FROM 'bcp_file.bcp'
  FORMAT BCP
  ...
  ```

For information on the LOAD TABLE ENCRYPTED clause, see *Advanced Security in Sybase IQ*.

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 The STRIP clause specifies whether unquoted values should have trailing blanks stripped off before they are inserted. The LOAD TABLE command accepts the following STRIP keywords:

- **STRIP OFF** Do not strip off trailing blanks.
- **STRIP RTRIM** Strip trailing blanks.
- **STRIP ON** Deprecated. Equivalent to STRIP RTRIM.

With STRIP turned on (the default), Sybase IQ strips trailing blanks from values before inserting them. This is effective only for VARCHAR data. STRIP OFF preserves trailing blanks.

Trailing blanks are stripped only for unquoted strings. Quoted strings retain their trailing blanks. If you do not require blank sensitivity, you can use the FILLER option as an alternative to be more specific in the number of bytes to strip, instead of all the trailing spaces. STRIP OFF is more efficient for Sybase IQ, and it adheres to the ANSI standard when dealing with trailing blanks. (CHAR data is always padded, so the STRIP option only affects VARCHAR data.)

The STRIP option applies only to variable-length non-binary data and does not apply to ASCII fixed-width inserts. For example, assume the following schema:

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

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

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

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

Trailing blanks are always trimmed from binary 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, 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.

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.

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.

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 *System Administration Guide: Volume 1*.

Use the START ROW ID option for partial-width inserts only. If the columns being loaded already contain data, the insert fails.

The START ROW ID clause of the LOAD TABLE and the INSERT commands is not allowed on a partitioned table.

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 CHECK 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 or named pipes.
- 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 1-12.

Table 1-12: 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 *System Administration Guide: Volume 1*.

Error messages

Sybase IQ no longer returns an error message when FORMAT BCP is specified as a LOAD TABLE clause. In addition, the following conditions are verified and proper error messages are returned.

- If the specified load format is not ASCII, BINARY, or BCP, Sybase IQ now returns the message "`Only ASCII, BCP and BINARY are supported LOAD formats.`"
- If the LOAD TABLE column specification contains anything other than column name, NULL, or ENCRYPTED, then Sybase IQ returns the new error message "`Invalid load specification for LOAD ... FORMAT BCP.`"

- If the column delimiter or row terminator size for the FORMAT BCP load is greater than 10 characters, then Sybase IQ returns the message "`Delimiter '%2' must be 1 to %3 characters in length.`" (where %3 equals 10).

  Messages corresponding to error or warning conditions which can occur for FORMAT BCP as well as FORMAT ASCII are the same for both formats.

- If the load default value specified is AUTOINCREMENT, IDENTITY, or GLOBAL AUTOINCREMENT, the error "`Default value %2 cannot be used as a LOAD default value. %1`" is reported.

- If the LOAD TABLE specification does not contain any columns that need to be loaded from the file specified, the error "`The LOAD statement must contain at least one column to be loaded from input file.`" is reported and the LOAD TABLE statement rolls back.

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 7 in Chapter 1, "Running the Database Server" in the *Utility Guide*.

LOAD TABLE also requires an exclusive lock on the table.

See also

INSERT statement on page 216

"LOAD_ZEROLENGTH_ASNULL option" on page 409

"NON_ANSI_NULL_VARCHAR option" on page 425

"Bulk loading data using the LOAD TABLE statement" in Chapter 7, "Moving Data In and Out of Databases" in the *System Administration Guide: Volume 1*

"Monitoring disk space usage," Chapter 14, "Troubleshooting Hints," in the *System Administration Guide: Volume 1*

LOCK TABLE statement

Description

Prevents other concurrent transactions from accessing or modifying a table within the specified time.

Syntax

LOCK TABLE *table-list* [**WITH HOLD**] **IN** { **SHARE** | **WRITE** | **EXCLUSIVE** } **MODE** [**WAIT** *time*]

Parameters

table-list:
[*owner.*] *table-name* [, [*owner.*] *table-name*, …]

time:

string

Examples

Example 1 For example, the following statement obtains a WRITE lock on the Customers and Employees tables, if available within 5 minutes and 3 seconds:

```
LOCK TABLE Customers, Employees IN WRITE MODE WAIT
'00:05:03'
```

Example 2 The following statement waits indefinitely, until the WRITE lock on the Customers and Employees tables, if available or an interrupt occurs:

```
LOCK TABLE Customers, Employees IN WRITE MODE WAIT
```

Usage

table-name The table must be a base table, not a view. WRITE mode is only valid for IQ base tables. LOCK TABLE either locks all tables in the table list, or none. If obtaining a lock for a SQL Anywhere table, or when obtaining SHARE or EXCLUSIVE locks, you may only specify a single table. Standard Sybase IQ object qualification rules are used to parse *table-name*. For related details, see the section "Identifiers" in Chapter 2, "SQL Language Elements" in *Reference: Building Blocks, Tables, and Procedures* and "Types of tables" in Chapter 5, "Working with Database Objects" in the *System Administration Guide: Volume 1*.

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.

WRITE Prevents other transactions from modifying a list of tables. Unconditionally commits the connections outermost transaction.The transaction's snapshot version is established not by the LOCK TABLE IN WRITE MODE statement, but by the execution of the next command processed by Sybase IQ.

A WRITE mode lock on an IQ table that participates in a join index also locks:

- The top table of the join index hierarchy in WRITE mode when X is a non-top table
- The corresponding join virtual table (JVT)

WRITE mode locks are released when the transaction commits or rolls back, or when the connection disconnects.

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 not to acquire row locks for *t*. This behavior can be disabled by setting the SUBSUME_ROW_LOCKS option OFF. For more information on the SUBSUME_ROW_LOCKS option, refer to "subsume_row_locks option [database]" in *SQL Anywhere Server – Database Administration > Configuring Your Database > Database options > Introduction to database options > Alphabetical list of options*.

LOCK TABLE statements run on tables in the IQ main store on the coordinator do not affect access to those tables from connections on secondary servers. For example:

On a coordinator connection, issue the command:

```
LOCK TABLE coord1 WITH HOLD IN EXCLUSIVE MODE
```

sp_iqlocks on the coordinator confirms that the table coord1 has an exclusive (E) lock.

The result of sp_iqlocks run on a connection on a secondary server does not show the exclusive lock on table coord1. The user on this connection can see updates to table coord1 on the coordinator.

Other connections on the coordinator can see the exclusive lock on coord1 and attempting to select from table coord1 from another connection on the coordinator returns the error `"User DBA has the row in coord1 locked."`

WAIT time Wait options specify maximum blocking time for all lock types. This option is mandatory when lock mode is WRITE. When a time argument is given, the server locks the specified tables only if available within the specified time. The time argument can be specified in the format *hh:nn:ss:sss*. If a date part is specified, the server ignores it and converts the argument into a timestamp. When no time argument is given, the server waits indefinitely until a WRITE lock is available or an interrupt occurs.

LOCK TABLE on views is unsupported. Attempting to lock a view acquires a shared schema lock regardless of the mode specified in the command. A shared schema lock prevents other transactions from modifying the table schema.

The Transact-SQL (TSQL) stored procedure dialect does not support LOCK TABLE. For example, the following statement returns `Syntax error near LOCK`:

```
CREATE PROCEDURE tproc()
AS
BEGIN
COMMIT
LOCK TABLE t1 IN SHARE MODE
INSERT INTO t1 VALUES(30)
END
```

The Watcom-SQL stored procedure dialect supports LOCK TABLE. The default command delimiter is a semicolon (;). For example:

```
CREATE PROCEDURE wproc()
BEGIN
COMMIT;
LOCK TABLE t1 IN SHARE MODE;
INSERT INTO t1 VALUES (20);
END
```

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 SQL 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 291

sp_iqlocks procedure in Chapter 7, "System Procedures," in *Reference: Building Blocks, Tables, and Procedures*

LOOP statement

Description

Repeats the execution of a statement list.

Syntax

[*statement-label*:]
... [**WHILE** *search-condition*] **LOOP**
... *statement-list*
... **END LOOP** [*statement-label*]

Examples

Example 1 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 ;
...
```

Example 2 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 197

LEAVE statement on page 229

MESSAGE statement

Description

Displays a message.

Syntax

MESSAGE *expression*, …
[**TYPE** { **INFO** | **ACTION** | **WARNING** | **STATUS** }]
[**TO** { **CONSOLE**
| **CLIENT** [**FOR** { **CONNECTION** *conn_id* [**IMMEDIATE**] | **ALL** }]
| [**EVENT** | **SYSTEM**] **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.
- **LOG** – 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

Example 1 Displays the string `The current date and time`, and the current date and time, on the database server message window:

```
CREATE PROCEDURE message_test ()
BEGIN
MESSAGE 'The current date and time: ', Now();
END;
CALL message_test();
```

Example 2 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.

For information about the IMMEDIATE parameter, see “MESSAGE statement” in *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (E-O)*.

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 SQL 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 120

“DEBUG_MESSAGES option” on page 375

“Using callback functions” in *SQL Anywhere Server – Programming > SQL Anywhere Database Tools Interface > Database tools interface > Using the database tools interface* 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

Example 1 Examples showing the use of OPEN in Embedded SQL:

```
EXEC SQL OPEN employee_cursor;
```

and

```
EXEC SQL PREPARE emp_stat FROM
'SELECT EmployeeID, Surname FROM Employees WHERE name
like ?';
EXEC SQL DECLARE employee_cursor CURSOR FOR emp_stat;
EXEC SQL OPEN employee_cursor USING :pattern;
```

Example 2 An example from a procedure:

```
BEGIN
DECLARE cur_employee CURSOR FOR
   SELECT Surname
   FROM Employees ;
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 1, “Using Procedures and Batches” in the *System Administration Guide: Volume 2*.

A cursor declared FOR READ ONLY sees the version of table(s) on which the cursor is declared when the cursor is opened, not the version of table(s) at the time of the first FETCH

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 59

DECLARE CURSOR statement [ESQL] [SP] on page 159

FETCH statement [ESQL] [SP] on page 193

PREPARE statement [ESQL] on page 268

RESUME statement on page 284

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 Employees table in a file in ASCII format:

```
SELECT * FROM Employees;
OUTPUT TO employee.txt FORMAT ASCII
```

Example 2 Places the contents of the Employees table at the end of an existing file, and includes any messages about the query in this file as well:

```
SELECT * FROM Employees;
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 HEXADECIMAL 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 line feed 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 379

SELECT statement on page 291

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 Surname
FROM Employees
WHERE DepartmentID = {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 275

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 Surname FROM Employees';
```

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 159

DESCRIBE statement [ESQL] on page 173

DROP STATEMENT statement [ESQL] on page 184

EXECUTE statement [ESQL] on page 186

OPEN statement [ESQL] [SP] on page 260

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; execute these statements inside a procedure:

```
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 411 or "sp_iqmodifylogin procedure" in Chapter 7, "System Procedures" in *Reference: Building Blocks, Tables, and Procedures*.

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 257

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 :EmployeeID, :Surname;
```

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 SET statement [ESQL].

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 “Value-sensitive cursors” in *SQL Anywhere Server – Programming > Introduction to Programming with SQL Anywhere > Using SQL in applications > SQL Anywhere cursors*.

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 171

INSERT statement on page 216

UPDATE statement on page 322

UPDATE (positioned) statement [ESQL] [SP] on page 326

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 361

ON_TSQL_ERROR option [TSQL] on page 427

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 267). 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 "READ statement [Interactive SQL]" in *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (P-Z)*.

Side effects

None.

Standards

- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions

None.

See also

DEFAULT_ISQL_ENCODING option [DBISQL] on page 379

PARAMETERS statement [DBISQL] on page 267

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 1, "Using Procedures and Batches" in the *System Administration Guide: Volume 2*. 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 290

SAVEPOINT statement on page 291

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 47

SIGNAL statement on page 312

RESTORE statement

Description

Restores a Sybase IQ database backup from one or more archive devices.

Syntax

Syntax 1

RESTORE DATABASE '*db_file*'
FROM '*archive_device*' [**FROM** '*archive_device*']…
… [**KEY** *key_spec*]
… [**RENAME** *dbspace-name* **TO** '*new-dbspace-path*']…
… [**CATALOG ONLY**]

Syntax 2

RESTORE DATABASE '*database-name*'
[*restore-option* ...]
FROM '*archive_device*' ...

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.

restore-option:

READONLY *dbspace-or-file* [, …]
KEY *key_spec*
RENAME *file-name* **TO** *new-file-path* ...

Examples

Example 1 The following UNIX example restores the iqdemo 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 'iqdemo'
FROM '/dev/rmt/0n'
FROM '/dev/rmt/2n'
```

Example 2 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'
```

Example 3 The following example shows the syntax of a BACKUP statement and two possible RESTORE statements.

Given the following BACKUP statement:

```
BACKUP DATABASE READONLY DBSPACES iq_main
TO '/system1/IQ15/IQ-15_1/demo/backup/iqmain'
```

The dbspace iq_main can be restored using either of the following RESTORE statements:

```
RESTORE DATABASE 'iqdemo' READONLY DBSPACES iq_main
FROM '/system1/IQ15/IQ-15_0/demo/backup/iqmain'
```

or

```
RESTORE DATABASE 'iqdemo'
FROM '/system1/IQ15/IQ-15_0/demo/backup/iqmain'
```

A selective backup backs up either all READWRITE dbspaces or specific read-only dbspaces or dbfiles.

Notes:

- You can take a READONLY selective backup and restore all objects from this backup (as in the second example above).
    - You can take an all-inclusive backup and restore read-only files and dbspaces selectively.
- You can take a READONLY selective backup of multiple read-only files and dbspaces and restore a sub-set of real-only files and dbspaces selectively.
    - You can restore the read-only backup, only if the read-only files have not changed since the backup. Once the dbspace is made read-write again, the read-only backup is invalid, unless you restore the entire read-write portion of the database back to the point at which the read-only dbspace was read-only.

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.

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 12, “Data Backup, Recovery, and Archiving,” in the *System Administration Guide: Volume 1*.

BACKUP allows you to specify full or incremental backups. There are 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.

To restore read-only files or dbspaces from an archive backup, the database may be running and the administrator may connect to the database when issuing the RESTORE statement. The read-only file pathname need not match the names in the backup, if they otherwise match the database system table information.

The database must not be running to restore a FULL, INCREMENTAL SINCE FULL, or INCREMENTAL restore of either a READWRITE FILES ONLY or an all files backup. The database may or may not be running to restore a backup of read-only files. When restoring specific files in a read-only dbspace, the dbspace must be offline. When restoring read-only files in a read-write dbspace, the dbspace can be online or offline. The restore closes the read-only files, restores the files, and reopens those files at the end of the restore.

You can use selective restore to restore a read-only dbspace, as long as the dbspace is still in the same read-only state.

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 *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 *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 41

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

Example 1 Embedded SQL examples:

```
EXEC SQL RESUME cur_employee;
```

and

```
EXEC SQL RESUME :cursor_var;
```

Example 2 dbisql example:

```
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 159

RETURN statement

Description

Exits a function or procedure unconditionally, optionally providing a return value. Statements following RETURN are not executed.

Syntax

RETURN [(*expression*)]

Examples

Example 1 Returns the product of three numbers:

```
CREATE FUNCTION product ( a numeric,
                          b numeric ,
                          c numeric)
RETURNS numeric
BEGIN
   RETURN ( a * b * c ) ;
END
```

Example 2 Calculates the product of three numbers:

```
SELECT product (2, 3, 4)
```

```
product (2,3,4)
24
```

Example 3 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 Customers)
   OR customer_id IS NULL THEN
      RETURN
   ELSE
      SELECT ID,sum(
         SalesOrderItems.Quantity )
      FROM Products,
            SalesOrderItems,
            SalesOrders
      WHERE SalesOrders.CustomerID = customer_id
      AND SalesOrders.ID = SalesOrderItems.ID
      AND SalesOrderItems.ProductID = Products.D
      GROUP BY Products.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 47

CREATE PROCEDURE statement on page 120

REVOKE statement

Description

Removes permissions for specified users.

Syntax

Syntax 1

REVOKE
{ **CONNECT** | **DBA** | **INTEGRATED LOGIN** | **GROUP**
| **KERBEROS LOGIN** | **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* [, …]

Syntax 4

REVOKE CREATE ON *dbspace-name* **FROM** *userid* [, …]

Examples

Example 1 Prevents user "dave" from inserting into the Employees table:

```
REVOKE INSERT ON Employees FROM dave ;
```

Example 2 Revokes resource permission from user "Jim":

```
REVOKE RESOURCE FROM Jim ;
```

Example 3 Prevents user "dave" from updating the Employees table:

```
REVOKE UPDATE ON Employees FROM dave ;
```

Example 4 Revokes integrated login mapping from the user profile name "Administrator":

```
REVOKE INTEGRATED LOGIN FROM Administrator ;
```

Example 5 Disallows the finance group from executing the procedure sp_customer_list:

```
REVOKE EXECUTE ON sp_customer_list
FROM finance ;
```

Example 6 Drops user ID franw from the database:

```
REVOKE CONNECT FROM franw ;
```

Example 7 Revokes CREATE privilege on dbspace DspHist from user Latifah:

```
REVOKE CREATE ON DspHist FROM Latifah
```

Example 8 Revokes CREATE permission on dbspace *DspHist* from user ID fionat from the database:

```
REVOKE CREATE ON DspHist FROM fionat ;
```

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.

REVOKE CREATE removes Create permission on the specified dbspace from the specified user IDs.

You cannot revoke permissions for a specific user within a group. If you do not want a specific user to access a particular table, view, or procedure, then do not make that user a member of a group that has permissions on that object.

Note You cannot revoke the connect privileges of a user 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.

For Syntax 4, you must have DBA authority.

See also

GRANT statement on page 206

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 62

ROLLBACK TO SAVEPOINT statement on page 290

ROLLBACK TO SAVEPOINT statement

Description

Cancels any changes made since a SAVEPOINT.

Syntax

ROLLBACK TO SAVEPOINT [*savepoint-name*]

Usage

The ROLLBACK TO SAVEPOINT statement will undo any changes that have been made since the SAVEPOINT was established. Changes made prior to the SAVEPOINT are not undone; they are still pending. For a description of savepoints, see Chapter 1, "Using Procedures and Batches" in the *System Administration Guide: Volume 2*.

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 276

ROLLBACK statement on page 289

SAVEPOINT statement on page 291

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 1, “Using Procedures and Batches” in the *System Administration Guide: Volume 2.*

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 276

ROLLBACK TO SAVEPOINT statement on page 290

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

Example 1 Lists all the tables and views in the system catalog:

```
SELECT tname
FROM SYS.SYSCATALOG
WHERE tname LIKE 'SYS%' ;
```

Example 2 Lists all customers and the total value of their orders:

```
SELECT CompanyName,
   CAST( sum(SalesOrderItems.Quantity *
   Products.UnitPrice) AS INTEGER) VALUE
FROM Customers
   LEFT OUTER JOIN SalesOrders
   LEFT OUTER JOIN SalesOrderItems
   LEFT OUTER JOIN Products
GROUP BY CompanyName
ORDER BY VALUE DESC
```

Example 3 Lists the number of employees:

```
SELECT count(*)
FROM Employees;
```

Example 4 Shows an Embedded SQL SELECT statement:

```
SELECT count(*) INTO :size FROM Employees;
```

Example 5 Lists the total sales by year, model, and color:

```
SELECT year, model, color, sum(sales)
FROM sales_tab
GROUP BY ROLLUP (year, model, color);
```

Example 6 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.

In SELECT statements, a stored procedure call can appear anywhere a base table or view is allowed. Note that CIS functional compensation performance considerations apply. For example, a SELECT statement can also return a result set from a procedure. For syntax and an example, see “FROM clause” in the *SQL Anywhere Server – SQL Reference > Using SQL > SQL statements > SQL statements (E-O)*. See “Creating and selecting from temporary tables” in Chapter 1, “Using Procedures and Batches” in the *System Administration Guide: Volume 2* 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 – 2147483647 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. If both TOP and ROW_COUNT are set, then the value of TOP takes precedence.

The ROW_COUNT option could produce inconsistent results when used in a query involving global variables, system functions or proxy tables. See "ROW_COUNT option" on page 442 for details.

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 4, “SQL Functions” in *Reference: Building Blocks, Tables, and Procedures*.

Note In Sybase IQ, scalar subqueries (nested selects) are allowed in the select list of the top level SELECT, as in SQL 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” and “Subqueries in search conditions” in Chapter 2, “SQL Language Elements” in *Reference: Building Blocks, Tables, and Procedures*.

---

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. If you use the same name or expression for a column alias as the column name, the name is processed as an aliased column, not a table column name.

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.

A SELECT INTO from a stored procedure or function is not permitted, as SELECT INTO is an atomic statement and you cannot do COMMIT, ROLLBACK, or some ROLLBACK TO SAVEPOINT statements in an atomic statement. For more information, see “Atomic compound statements” and “Transactions and savepoints in procedures” in Chapter 1, “Using Procedures and Batches” of the *System Administration Guide: Volume 2*.

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 SQL 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 200. 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 SQL 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 SQL Anywhere documentation for rules that might apply to processing.

If you have a query that does not require a FROM clause, you can force 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” in Chapter 2, “SQL Language Elements” in *Reference: Building Blocks, Tables, and Procedures* for a full description.

Sybase IQ also supports the disjunction of subquery predicates. Each subquery can appear within the WHERE or HAVING clause with other predicates and can be combined using the AND or OR operators. See “Disjunction of subquery predicates” in Chapter 2, “SQL Language Elements” in *Reference: Building Blocks, Tables, and Procedures*.

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 constant, 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” in Chapter 2, “SQL Language Elements” in *Reference: Building Blocks, Tables, and Procedures* for the format of an operator expression.

GROUPING takes a column name as a parameter and returns a Boolean value as listed in Table 1-13.

Table 1-13: 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 2, "Using OLAP," in the *System Administration Guide: Volume 2*.

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 1-14.

Table 1-14: 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 2, "Using OLAP," in the *System Administration Guide: Volume 2*.

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 155

DECLARE CURSOR statement [ESQL] [SP] on page 159

FETCH statement [ESQL] [SP] on page 193

FROM clause on page 200

OPEN statement [ESQL] [SP] on page 260

UNION operation on page 321

"SUBQUERY_CACHING_PREFERENCE option" on page 447

"Search conditions," "Disjunction of subquery predicates," and "Expressions" in Chapter 2, "SQL Language Elements" in *Reference: Building Blocks, Tables, and Procedures*

Chapter 2, "Using OLAP," in *System Administration Guide: Volume 2*

"Accessing fields and methods of the Java object" in *SQL Anywhere Server – Programming > Java in the database > Java support in SQL Anywhere*

SET statement [ESQL]

Description

Assigns a value to a SQL variable.

Syntax

SET *identifier* = *expression*

Examples

Example 1 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;
```

Example 2 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 153

DROP VARIABLE statement on page 186

"Expressions" in Chapter 2, "SQL Language Elements" in *Reference: Building Blocks, Tables, and Procedures*

SET statement [T-SQL]

Description

Sets database options in an Adaptive Server Enterprise-compatible manner.

Syntax

SET *option-name option-value*

Usage

Table 1-15 lists available options.

Table 1-15: Transact-SQL SET options

Option name	Option value
ANSINULL	ON | OFF
ANSI_PERMISSIONS	ON | OFF
CLOSE_ON_ENDTRANS	ON
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 }** When CLOSE_ON_ENDTRANS is set to ON (the default and only allowable value), cursors are closed at the end of a transaction. With the option set ON, CLOSE_ON_ENDTRANS 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 *System Administration Guide: Volume 1*. 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 307

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
Example 1 In Embedded SQL:

```
EXEC SQL SET CONNECTION :conn_name
```

Example 2 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 65

DISCONNECT statement [DBISQL] on page 176

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 4.

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 156

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

Example 1 Sets the DATE_FORMAT option:

```
SET OPTION public.date_format = 'Mmm dd yyyy'
```

Example 2 Sets the WAIT_FOR_COMMIT option to on:

```
SET OPTION wait_for_commit = 'on'
```

Example 3 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 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 following 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 SQL Communication Area (SQLCA)" in *SQL Anywhere Server – Programming > SQL Anywhere Data Access APIs > SQL Anywhere embedded SQL*

SIGNAL statement

Description

Signals an exception condition.

Syntax

SIGNAL *exception-name*

Usage

SIGNAL lets you raise an exception. See Chapter 1, "Using Procedures and Batches," in the *System Administration Guide: Volume 2* 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 47

RESIGNAL statement on page 278

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

Example 1 On a UNIX system, starts the database file */s1/sybase/sample_2.db* on the current server:

```
START DATABASE '/s1/sybase/sample_2.db'
```

Example 2 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\IQ-15_1\demo\iqdemo.db* is given the default name iqdemo.

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

Example 1 Start a database server, named “eng1”, without starting any databases on it:

```
START ENGINE AS eng1
```

Example 2 The following example shows the use of a STARTLINE clause.

```
START ENGINE AS eng1 STARTLINE 'start_iq -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 *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_iq command.

Side effects

None

Standards

- **SQL92** Vendor extension.
- **Sybase** Not applicable.

Permissions

None.

See also

STOP ENGINE statement [DBISQL] on page 317

Chapter 1, “Running the Database Server” in the *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 317

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 176

START DATABASE statement [DBISQL] on page 313

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 314

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 315

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 114

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 14

CREATE EVENT statement on page 86

Chapter 6, “Automating Tasks Using Schedules and Events” in the *System Administration Guide: Volume 2*

TRUNCATE TABLE statement

Description

Deletes all rows from a table without deleting the table definition.

Syntax

Syntax 1

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

Syntax 2

TRUNCATE TABLE [*owner* .]*table* [**PARTITION** *partition-name*]

Examples

Deletes all rows from the Sale table:

```
TRUNCATE TABLE Sale
```

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.

The partition clause specifies which partition to truncate. It does not affect data in other partitions.

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 SQL 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 169

Chapter 10, “Transactions and Versioning” in *System Administration Guide: Volume 1*

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 Surname
FROM Employees
UNION
SELECT Surname
FROM Customers
```

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. See “FROM clause” on page 200.

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.

Note When SELECT statements include constant values and UNION ALL views but omit the FROM clause, use iq_dummy to avoid errors. See “FROM clause” on page 200 for details.

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 291

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

Example 1 Transfers employee Philip Chin (employee 129) from the sales department to the marketing department:

```
UPDATE Employees
SET DepartmentID = 400
WHERE EmployeeID = 129;
```

Example 2 The Marketing Department (400) increases bonuses from 4% to 6% of each employee's base salary:

```
UPDATE Employees
SET bonus = base * 6/100
WHERE DepartmentID =400;
```

Example 3 Each employee gets a pay increase with the department bonus:

```
UPDATE Employees
SET emp.Salary = emp.Salary + dept.bonus
FROM Employees emp, Departments dept
WHERE emp.DepartmentID = dept.DepartmentID;
```

Example 4 Another way to give each employee a pay increase with the department bonus:

```
UPDATE Employees
SET emp.salary = emp.salary + dept.bonus
FROM Employees emp JOIN Departments dept
ON emp.DepartmentID = dept.DepartmentID;
```

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 "NULLIF function [Miscellaneous]" in Chapter 4, "SQL Functions" in *Reference: Building Blocks, Tables, and Procedures*.

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 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. The IQ server may return results in any combination of lowercase and uppercase, so you cannot expect case sensitive results in a database that is case insensitive (CASE IGNORE). 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 *System Administration Guide: Volume 1*.

See CREATE TABLE statement on page 135 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 4, “Accessing Remote Data” and Chapter 5, “Server Classes for Remote Data Access” in the *System Administration Guide: Volume 2*.

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 Employees SET surname = '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 SQL Anywhere.

Permissions

Must have UPDATE permission on the columns being modified.

See also

DECLARE CURSOR statement [ESQL] [SP] on page 159

DELETE statement on page 169

DELETE (positioned) statement [ESQL] [SP] on page 171

UPDATE statement on page 322

sp_iqcursorinfo procedure in Chapter 7, “System Procedures” in *Reference: Building Blocks, Tables, and Procedures*

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* }
[**CHECK EVERY** *integer* }
[**AFTER MESSAGE BREAK**]

Parameters

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

The WAITFOR statement wakes up periodically (every 5 seconds by default) to check if it has been canceled or if messages have been received. If neither of these has happened, the statement continues to wait.

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.

CHECK EVERY clause This optional clause controls how often the WAITFOR statement wakes up. By default, WAITFOR wakes up every 5 seconds. The value is in milliseconds, and the minimum value is 250milliseconds.

AFTER MESSAGE BREAK clause The WAITFOR statement can be used to wait for a message from another connection. In most cases, when a message is received it is forwarded to the application that executed the WAITFOR statement and the WAITFOR statement continues to wait. If the AFTER MESSAGE BREAK clause is specified, when a message is received from another connection, the WAITFOR statement completes. The message text is not forwarded to the application, but it can be accessed by obtaining the value of the MessageReceived connection property.

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 86

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 Products) < 30
BEGIN
    DELETE FROM Products
    WHERE UnitPrice = MAX(UnitPrice)
    IF ( SELECT MAX(UnitPrice) FROM Products ) < 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 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 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 307.

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 SQL Anywhere database options. Sybase IQ modifies some SQL Anywhere option defaults, and these modified values become the new default values. Unless the new Sybase IQ default value is changed again, sp_iqcheckoptions does not list the option.

  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 in Chapter 7, “System Procedures”in *Reference: Building Blocks, Tables, and Procedures*.

The system table DBA.SYSOPTIONDEFAULTS contains all of the names and default values of the Sybase IQ and SQL Anywhere 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
```

- 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

OS_FILE_CACHE_BUFFERING

OUT_OF_DISK_MESSAGE_REPEAT

OUT_OF_DISK_WAIT_TIME

PREFETCH_BUFFER_LIMIT

PREFETCH_BUFFER_PERCENT

RECOVERY_TIME

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 TDS (tabular data stream) 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, which is called after all the checks have been performed to verify that the connection is valid. The LOGIN_PROCEDURE option names a stored procedure to run when users connect. The default setting is to use the sp_login_environment system stored procedure. You can specify a different stored procedure.

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

For more information, see “LOGIN_PROCEDURE option” on page 411, or “sp_login_environment system procedure” and “sp_tsql_environment system procedure” in Chapter 7, “System Procedures” in *Reference: Building Blocks, Tables, and Procedures*.

Deprecated database options

See Chapter 2, “Behavior Changes in Sybase IQ 15.0” in *New Features in Sybase IQ 15.0* for information about database options deprecated in this release.

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 345 and “DBISQL options” on page 347 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
ALLOW_READ_CLIENT_FILE	ON, OFF	OFF
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
BTREE_PAGE_SPLIT_PAD_PERCENT	0 - 90	50
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_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
DBCC_PINNABLE_CACHE_PERCENT	0 – 100	50
DEBUG_MESSAGES	ON, OFF	OFF
DEFAULT_DBSPACE	string	'' (empty string)
DEFAULT_DISK_STRIPING	ON, OFF	ON
DEDICATED_TASK	ON, OFF	OFF
DEFAULT_HAVING_SELECTIVITY_PPM	0 – 1000000	0
DEFAULT_KB_PER_STRIPE	1 – max unsigned bigint	1
DEFAULT_LIKE_MATCH_SELECTIVITY_PPM	0 – 1000000	150000
DEFAULT_LIKE_RANGE_SELECTIVITY_PPM	1 – 1000000	150000
DELAYED_COMMIT_TIMEOUT	integer	500
DELAYED_COMMITS	OFF	OFF
DISABLE_RI_CHECK	ON, OFF	OFF
EARLY_PREDICATE_EXECUTION	ON, OFF	ON
EXTENDED_JOIN_SYNTAX	ON, OFF	ON

OPTION	VALUES	DEFAULT
FORCE_DROP	ON, OFF	OFF
FORCE_NO_SCROLL_CURSORS	ON, OFF	OFF
FORCE_UPDATABLE_CURSORS	ON, OFF	OFF
FP_LOOKUP_SIZE	1 MB – 4096 MB	16 MB
FP_LOOKUP_SIZE_PPM	1 – 1000000	2500
FP_PREDICATE_WORKUNIT_PAGES	integer	200
FP_PREFETCH_SIZE	0 – 100	10
FPL_EXPRESSION_MEMORY_KB	0 – 20000	1024
GARRAY_FILL_FACTOR_PERCENT	0 – 1000	25
GARRAY_INSERT_PREFETCH_SIZE	0 – 100	3
GARRAY_PAGE_SPLIT_PAD_PERCENT	0-100	25
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	ON
IN_SUBQUERY_PREFERENCE	-3 – 3	0
IQGOVERN_MAX_PRIORITY	1 – 3	2
IQGOVERN_PRIORITY	1 – 3	2
IQGOVERN_PRIORITY_TIME	1 – 1000000 seconds	0 (disabled)
ISOLATION_LEVEL	0, 1, 2, 3	0
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
LOAD_ZEROLENGTH_ASNULL	ON, OFF	OFF
LOCKED	ON, OFF	OFF
LOG_CONNECT	ON, OFF	ON
LOG_CURSOR_OPERATIONS	ON, OFF	OFF

OPTION	VALUES	DEFAULT
LOGIN_MODE	STANDARD, MIXED, INTEGRATED	STANDARD
LOGIN_PROCEDURE	string	sp_login_environment
MAIN_RESERVED_DBSPACE_MB	integer >= 200 in MB	200
MAX_CARTESIAN_RESULT	integer	100000000
MAX_CLIENT_NUMERIC_PRECISION	0 – 126	0
MAX_CLIENT_NUMERIC_SCALE	0 – 126	0
MAX_CONNECTIONS	0 - 2147483647	Unlimited
MAX_CUBE_RESULT	0 – 4294967295	10000000
MAX_CURSOR_COUNT	integer	50
MAX_DAYS_SINCE_LOGIN	0 - 2147483647	Unlimited
MAX_FAILED_LOGIN_ATTEMPTS	0 - 2147483647	Unlimited
MAX_HASH_ROWS	integer to 4294967295	2500000
MAX_IQ_THREADS_PER_CONNECTION	3 – 10000	144
MAX_IQ_THREADS_PER_TEAM	1 – 10000	144
MAX_JOIN_ENUMERATION	1 – 64	15
MAX_NON_DBA_CONNECTIONS	0 - 2147483647	Unlimited
MAX_QUERY_PARALLELISM	integer <= # CPUs	24
MAX_QUERY_TIME	$0 – 2^{32} - 1$	0 (disabled)
MAX_STATEMENT_COUNT	integer	100
MAX_TEMP_SPACE_PER_CONNECTION	integer	0
MAX_WARNINGS	integer	$2^{48} - 1$
MINIMIZE_STORAGE	ON, OFF	OFF
MIN_PASSWORD_LENGTH	integer >= 0	0 characters
MONITOR_OUTPUT_DIRECTORY	string	*database directory*
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
PASSWORD_GRACE_TIME	0 - 2147483647	0
PASSWORD_EXPIRY_ON_NEXT_LOGIN	ON, OFF	OFF
PASSWORD_LIFE_TIME	0 - 2147483647	Unlimited
POST_LOGIN_PROCEDURE	string	sp_iq_process_post_login
PRECISION	126	126

OPTION	VALUES	DEFAULT
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	20
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_PLAN_TEXT_ACCESS	ON, OFF	OFF
QUERY_PLAN_TEXT_CACHING	ON, OFF	OFF
QUERY_ROWS_RETURNED_LIMIT	integer	0
QUERY_TEMP_SPACE_LIMIT	integer	0
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_COLLATION	Internal, collation_name, or collation_id	Internal
SORT_PINNABLE_CACHE_PERCENT*	0 – 100	20
SUBQUERY_CACHING_PREFERENCE	-3 – 3	0
SUBQUERY_FLATTENING_PERCENT	0, 1 - 2^{32} -1	100
SUBQUERY_FLATTENING_PREFERENCE	-3 – 3	0
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_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	','

OPTION	VALUES	DEFAULT
TEMP_EXTRACT_DIRECTORY	string	'' (empty string)
TEMP_EXTRACT_ESCAPE_QUOTES	ON, OFF	OFF
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_RESERVED_DBSPACE_MB	integer >= 200 in MB	200
TEMP_SPACE_LIMIT_CHECK	ON, OFF	ON
TIME_FORMAT	string	'HH:NN:SS.SSS'
TIMESTAMP_FORMAT	string	'YYYY-MM-DD HH:NN:SS.SSS'
TOP_NSORT_CUTOFF_PAGES	1 – 1000	1
TRIM_PARTIAL_MBC	ON, OFF	OFF
USER_RESOURCE_RESERVATION	integer	1
VERIFY_PASSWORD_FUNCTION	string	'' (empty string)
WASH_AREA_BUFFERS_PERCENT	1 – 100	20
WAIT_FOR_COMMIT	ON, OFF	OFF
WD_DELETE_METHOD	0 – 3	0

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 *System Administration Guide: Volume 1*.

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 [ESQL] on page 301.

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 *Reference: Building Blocks, Tables, and 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
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_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 339 and “DBISQL options” on page 347 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*	ON, OFF	OFF
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	OFF
ASE_FUNCTION_BEHAVIOR	ON, OFF	OFF
CHAINED	ON, OFF	ON
CLOSE_ON_ENDTRANS	ON	ON
CONTINUE_AFTER_RAISERROR	ON, OFF	ON
CONVERSION_ERROR	ON, OFF	ON
ESCAPE_CHARACTER*	Reserved	Reserved
FIRE_TRIGGERS*	ON, OFF	ON
NEAREST_CENTURY	0 – 100	50
NON_KEYWORDS	Comma-separated keywords list	No keywords turned off
ON_TSQL_ERROR	STOP, CONTINUE, CONDITIONAL	CONDITIONAL
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	ON
TEXTSIZE*		
TSQL_HEX_CONSTANT*		
TSQL_VARIABLES	ON, OFF	OFF

Notes
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 [**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*

Description

Syntax 1 with the TEMPORARY keyword cannot be used between the BEGIN and END keywords of a compound statement.

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 339 and "Transact-SQL compatibility options" on page 345 for lists of the other classes of options.

Table 2-4: DBISQL options

Option	Values	Default
DEFAULT_ISQL_ENCODING	Identifier or string	empty string (use system code page)
NULLS*	String	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*	Non-negative integer	0 (no truncation)

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 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

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” in *Reference: Building Blocks, Tables, and Procedures*

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 SQL 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

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 322

ALLOW_READ_CLIENT_FILE option

Function

Enables client-side data transfer. For details, see “allow_read_client_file option [database]” in *SQL Anywhere Server – Database Administration > Configuring Your Database > Introduction to database options > Alphabetical list of options*.

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.

The APPEND_LOAD database option behaves differently for partitioned and non-partitioned tables. Row ID ranges are assigned to each partition in a partitioned table. For partitioned tables, when APPEND_LOAD is ON, new rows are appended at the end of the appropriate partition. When APPEND_LOAD is OFF, the load reuses the first available row IDs and space from deleted rows.

For non-partitioned tables, when APPEND_LOAD is ON, new rows are added after the maximum row ID that is at the end of the table rows. When APPEND_LOAD is OFF, the load reuses the deleted row IDs. With non-partitioned tables, you can also control where rows are inserted by using the LOAD or INSERT START ROW ID clause to specify the row at which to start inserting.

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
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 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 SQL 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 on binary data types. Sybase IQ supports bulk loading of remote data via the LOAD TABLE USING CLIENT FILE statement.

See also

LOAD TABLE statement on page 230

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]” and “HEXTOINT function [Data type conversion]” in Chapter 4, “SQL Functions” in *Reference: Building Blocks, Tables, and Procedures*.

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]" and "INTTOHEX function [Data type conversion]" in Chapter 4, "SQL Functions" in *Reference: Building Blocks, Tables, and Procedures*

"CONVERSION_ERROR option [TSQL]" on page 362

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 details about many events in the database in the transaction log. Auditing provides some security features, at the cost of some performance. When you turn on auditing for a database, you cannot stop using the transaction log. You must turn auditing off before you turn off the transaction log. Databases with auditing on cannot be started in read-only mode.

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

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

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

Can be set for the PUBLIC group only. Takes effect immediately. DBA authority required.

See also — "sa_enable_auditing_type system procedure" in Chapter 7, "System Procedures" in *Reference: Building Blocks, Tables, and Procedures*

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 391

"SORT_PINNABLE_CACHE_PERCENT option" on page 445

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 357

"PREFETCH_BUFFER_LIMIT option" on page 431

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.

BTREE_PAGE_SPLIT_PAD_PERCENT option

Function
Determines per-page fill factor during page splits for B-Tree structures. B-Tree structures are used by the HG, LF, DT, TIME, and DTTM indexes. Splits of a B-Tree page try to leave the specified percentage empty to avoid splitting when new keys are inserted into the index.

Allowed values
0 – 90

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
Indexes reserve storage at the page level that can be allocated to new keys as additional data is inserted. Reserving space consumes additional disk space, but can help the performance of incremental inserts. If future plans include incremental inserts, and the new rows do not have values that are already present in the index, a nonzero value for the GARRAY_PAGE_SPLIT_PAD_PERCENT option may improve incremental insert performance.

If you do not plan to incrementally update the index, you can reduce the value of this option to save disk space.

See also
“GARRAY_FILL_FACTOR_PERCENT option” on page 389

“GARRAY_PAGE_SPLIT_PAD_PERCENT option” on page 390

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 “Starting the database server” in Chapter 1, “Running the Database Server” in the *Utility Guide*

“Managing lock contention,” Chapter 10, “Transactions and Versioning,” in the *System Administration Guide: Volume 1*

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 441 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 4, “Accessing Remote Data” in the *System Administration Guide: Volume 2*.

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.

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 427

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.

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 '/s1/mydata/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']"
```

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

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 *System Administration Guide: Volume 1*.

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

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 SQL 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 Chapter 1, “Selecting Data from Database Tables” in *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 (StartDate, ‘yy’) + ‘?’
FROM Employees;
```

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 333

"RETURN_DATE_TIME_AS_STRING option" on page 442

"TIME_FORMAT option" on page 466

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:
IQ_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 13, "System Recovery and Database Repair" in the *System Administration Guide: Volume 1*

"sp_iqcheckdb procedure" in Chapter 7, "System Procedures" in *Reference: Building Blocks, Tables, and Procedures*

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” in Chapter 7, “System Procedures” in *Reference: Building Blocks, Tables, and Procedures*

“Resource issues running sp_iqcheckdb” in Chapter 13, “System Recovery and Database Repair”in the *System Administration Guide: Volume 2*

Chapter 13, “System Recovery and Database Repair”in the *System Administration Guide: Volume 1*

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 257

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_DBSPACE option

Function
Changes the default dbspace where tables or join indexes are created. Allows the administrator to set the default dbspace for a group or user or allows a user to set the user's own default dbspace.

Allowed values
String containing a dbspace name

Default
'' (the empty string)

Scope
Can be set for an individual connection or PUBLIC group. Setting takes effect immediately. Requires DBA permissions to set the option for groups or users other than the current user. Takes effect immediately.

Description
When a table is created without specifying a dbspace, the dbspace named by this option setting is used for base tables and join indexes. If this option is not set or is set to the empty string, the IQ_SYSTEM_MAIN dbspace is used. If this option is set to a non-existent or read-only dbspace, the create statement returns an error for base tables and join indexes. These rules also apply to tables created implicitly via a SELECT INTO command.

IQ_SYSTEM_TEMP is always used for global temporary tables unless a table IN clause is used that specifies SYSTEM, in which case an SA global temporary table is created.

At database creation, the system dbspace, IQ_SYSTEM_MAIN, is created and is implied when the PUBLIC.DEFAULT_DBSPACE option setting is empty or explicitly set to IQ_SYSTEM_MAIN. Immediately after creating the database, Sybase recommends that the administrator create a second main dbspace, revoke CREATE privilege in dbspace IQ_SYSTEM_MAIN from PUBLIC, grant CREATE in dbspace for the new main dbspace to selected users or PUBLIC, and set PUBLIC.DEFAULT_DBSPACE to the new main dbspace.

For example:

```
CREATE DBSPACE user_main USING FILE user_main
'user_main1' SIZE 10000;
GRANT CREATE ON user_main TO PUBLIC;
REVOKE CREATE ON IQ_SYSTEM_MAIN FROM PUBLIC;
SET OPTION PUBLIC.DEFAULT_DBSPACE = 'user_main';
```

Example

In this example, CONNECT and RESOURCE privileges on all dbspaces are granted to users usrA and usrB, and each of these users is granted CREATE privilege on a particular dbspace:

```
GRANT CONNECT, RESOURCE TO usrA, usrB
   IDENTIFIED BY pwdA, pwdB;
GRANT CREATE ON dbsp1 TO usrA;
GRANT CREATE ON dbsp3 TO usrB;
SET OPTION “usrA”.default_dbspace = ‘dbsp1’;
SET OPTION “usrB”.default_dbspace = ‘dbsp3’;
SET OPTION “PUBLIC”.default_dbspace = dbsp2;

CREATE TABLE “DBA”.t1(c1 int, c2 int);
INSERT INTO t1 VALUES (1, 1);
INSERT INTO t1 VALUES (2, 2);
COMMIT;
```

UsrA connects:

```
CREATE TABLE “UsrA”.t1(c1 int, c2 int);
INSERT INTO t1 VALUES (1, 1);
INSERT INTO t1 VALUES (2, 2);
COMMIT;
```

UsrB connects:

```
CREATE TABLE “UsrB”.t1(c1 int, c2 int);
INSERT INTO t1 VALUES (1, 1);
INSERT INTO t1 VALUES (2, 2);
COMMIT;
```

DBA connects:

```
SELECT Object, DbspaceName, ObjSize
FROM sp_iqindexinfo();
```

sp_iqindexinfo result:

```
DBA.t1                              dbsp2       200k
DBA.t1.ASIQ_IDX_T730_C1_FP          dbsp2       288k
DBA.t1.ASIQ_IDX_T730_C2_FP          dbsp2       288k
usrA.t1                             dbsp1       200k
usrA.t1.ASIQ_IDX_T731_C1_FP         dbsp1       288k
usrA.t1.ASIQ_IDX_T731_C2_FP         dbsp1       288k
usrB.t1                             dbsp3       200k
usrB.t1.ASIQ_IDX_T732_C1_FP         dbsp3       288k
usrB.t1.ASIQ_IDX_T732_C2_FP         dbsp3       288k
```

See also

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

DEFAULT_DISK_STRIPING option

Function

Sets default disk striping value for all dbspaces.

Allowed values

ON, OFF

Default

ON

Scope

Can be set for the PUBLIC group only. Requires DBA permissions.

Description

By default, disk striping is ON for all dbspaces in the IQ main store. This option is used only by CREATE DBSPACE and defines the default striping value, if CREATE DBSPACE does not specify striping.

DEFAULT_HAVING_SELECTIVITY_PPM option

Function

Provides default selectivity estimates to the optimizer for most HAVING clauses in parts per million.

Allowed values

0 – 1000000

Default

0

Scope

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

Description

DEFAULT_HAVING_SELECTIVITY_PPM 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_PPM 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_PPM 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” in the “Search conditions” section in Chapter 2, “SQL Language Elements” in *Reference: Building Blocks, Tables, and Procedures*.

See also

Chapter 3, “Optimizing Queries and Deletions” in the *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 only be set as a temporary option, 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 list of supported code pages, see "*Supported and alternate collations" in SQL Anywhere Server – Database Administration > Configuring Your Database > International languages and character sets > Character set and collation reference information*.

Example

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

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

See also

READ statement [DBISQL] on page 275 and OUTPUT statement [DBISQL] on page 263

"Overview of character sets, encodings, and collations" in *SQL Anywhere Server – Database Administration > Configuring Your Database > International languages and character sets > Understanding character sets*

DEFAULT_KB_PER_STRIPE option

Function

Sets default size in KB for all dbspaces in the IQ main store.

Allowed values

1 to maximum integer

Default

1

Scope

Can be set for the PUBLIC group only. Requires DBA permissions.

Description

By default disk striping size is 1KB for all dbspaces in the IQ main store. This option is used only by CREATE DBSPACE and defines the default disk striping size for dbspaces in the IQ main store, if CREATE DBSPACE does not specify a stripe size.

DEFAULT_LIKE_MATCH_SELECTIVITY_PPM option

Function

Provides default selectivity estimates (in parts per million) to the optimizer for most LIKE predicates.

Allowed values

0 to 1000000

Default

150000

Scope

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

Description

DEFAULT_LIKE_MATCH_SELECTIVITY_PPM 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 an LF index or a 1- or 2- or 3-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" in the "Search conditions" section in Chapter 2, "SQL Language Elements" in *Reference: Building Blocks, Tables, and Procedures*.

See also

"DEFAULT_LIKE_RANGE_SELECTIVITY_PPM option" on page 381 and "FP_LOOKUP_SIZE option" on page 386.

"The Fast Projection (FP) default index type" in Chapter 6, "Using Sybase IQ Indexes" in *System Administration Guide: Volume 1*

"LIKE conditions" in Chapter 2, "SQL Language Elements" in *Reference: Building Blocks, Tables, and Procedures*

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

DEFAULT_LIKE_RANGE_SELECTIVITY_PPM option

Function

Provides default selectivity estimates (in parts per million) to the optimizer for leading constant LIKE predicates.

Allowed values

1 to 1000000

Default

150000

Scope

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

Description

DEFAULT_LIKE_RANGE_SELECTIVITY_PPM 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 an LF index or a 1- or 2- or 3-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 46 in *Reference: Building Blocks, Tables, and Procedures*.

See also

“DEFAULT_LIKE_MATCH_SELECTIVITY_PPM option” on page 380 and “FP_LOOKUP_SIZE option” on page 386

“The Fast Projection (FP) default index type” in Chapter 6, “Using Sybase IQ Indexes” in *System Administration Guide: Volume 1*

“LIKE conditions” in Chapter 2, “SQL Language Elements” in *Reference: Building Blocks, Tables, and Procedures*

Chapter 3, “Optimizing Queries and Deletions” in the *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.

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

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 )
```

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. 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 13, "System Recovery and Database Repair"in the *System Administration Guide: Volume 1*.

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.

FP_LOOKUP_SIZE option

The maximum number of lookup pages used in Sybase IQ.

Function
To control amount of cache allocated to the creation of Lookup FP indexes, particularly FP(3) Indexes.

Allowed values
1 MB – 4096 MB

Default
16 MB

Scope
DBA permissions are required to set this option. Can be set temporary for an individual connection or for the PUBLIC group. Takes effect immediately.

Description
Controls the maximum number of lookup pages. For further details, see “The Fast Projection (FP) default index type” in Chapter 6, “Using Sybase IQ Indexes” in *System Administration Guide: Volume 1.*

The FP_LOOKUP_SIZE option must be set public, so the allowed syntax is:

```
SET OPTION public.FP_LOOKUP_SIZE = 1
```

Other options

The following options now support 3-byte indexes:

- INDEX_ADVISOR
- MINIMIZE_STORAGE
- FP_LOOKUP_SIZE_PPM

Stored procedures

The following stored procedures now support 3-byte indexes:

- sp_iqcheckdb
- sp_iqcolumn
- sp_iqindexadvice
- sp_iqindexmetadata
- sp_iqindexsize
- sp_iqindex
- sp_iqindexfragmentation
- sp_iqrebuildindex
- sp_iqrowdensity

See also

"FP_LOOKUP_SIZE_PPM option" on page 387

Chapter 6, "Using Sybase IQ Indexes" in *System Administration Guide: Volume 1*

"MINIMIZE_STORAGE option" on page 421

FP_LOOKUP_SIZE_PPM option

Restrict FP lookup storage size in Sybase IQ to this parts-per-million value of main memory.

Function

To control amount of main cache allocated to the creation of Lookup FP indexes for all FP lookup indexes, but particularly for FP(3) indexes.

Allowed values

1 to 1000000

Default

2500

Scope

DBA permissions are required to set this option. Can be set temporary for an individual connection or for the PUBLIC group. Takes effect immediately.

Description — Controls the maximum number of lookup pages and restricts this number to a parts-per-million value of main memory, that is, the value of FP_LOOKUP_SIZE_PPM * size of main memory / 1,000,000, where the size of main memory is as specific by the -iqmc server startup parameter.

For further details, see “The Fast Projection (FP) default index type” in Chapter 6, “Using Sybase IQ Indexes” in *System Administration Guide: Volume 1*.

Other options — The following options now support 3-byte indexes:

- FP_LOOKUP_SIZE
- INDEX_ADVISOR
- MINIMIZE_STORAGE

See also — “FP_LOOKUP_SIZE option” on page 386

Chapter 6, “Using Sybase IQ Indexes” in *System Administration Guide: Volume 1*

“MINIMIZE_STORAGE option” on page 421

FP_PREDICATE_WORKUNIT_PAGES option

Function — Specifies degree of parallelism used in the default index.

Allowed values — Integer

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 — 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.

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.

GARRAY_FILL_FACTOR_PERCENT option

Function

Specifies the percent of space on each HG garray pages to reserve for future incremental inserts into existing groups. The garray tries to pad out each group to include a pad of empty space set by the value.This space is used for rows added to existing index groups.

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 equivalent values). Reserving space consumes additional disk space but can help the performance of incremental inserts into the HG index.

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 might improve incremental insert performance.

If you do not plan to incrementally update the index, you can reduce the values of this option to save disk space.

See also

“GARRAY_PAGE_SPLIT_PAD_PERCENT option” on page 390

“sp_iqindexfragmentation procedure,” Chapter 7, “System Procedures,” in *Reference: Building Blocks, Tables, and Procedures*

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_PAGE_SPLIT_PAD_PERCENT option

Function

Determines per-page fill factor during page splits on the garray and specifies the percent of space on each HG garray page to reserve for future incremental inserts. Splits of a garray page try to leave that percentage empty. This space is used for rows added to new index groups.

Allowed values

0 – 100

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 storage at the page level that can be allocated to new groups when additional rows are inserted. Reserving space consumes additional disk space but can help the performance of incremental inserts into the HG index.

If future plans include incremental inserts into an HG index, and the new rows do not have values that are already present in the index, a nonzero value for the GARRAY_PAGE_SPLIT_PAD_PERCENT option could improve incremental insert performance.

If you do not plan to incrementally update the index, you can reduce the values of this option to save disk space.

See also

"GARRAY_FILL_FACTOR_PERCENT option" on page 389

"sp_iqindexfragmentation procedure," Chapter 7, "System Procedures," in *Reference: Building Blocks, Tables, and Procedures*

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 356

"SORT_PINNABLE_CACHE_PERCENT option" on page 445

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 14, "Troubleshooting Hints," in the *System Administration Guide: Volume 1*.

Also see "HASH_PINNABLE_CACHE_PERCENT option" on page 391

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 *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.

This option effectively controls the amount of time the optimizer spends searching for the best index to use for a range predicate. Setting this option higher may cause a query to spend more time in the optimizer, but as a result may choose a better index to resolve a range predicate.

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 434

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.

Note If you set a user level option for the current option, the corresponding temporary option is also set. For details, see "Scope and duration of database options" on page 335.

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.

Examples
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 = ''
```

To illustrate the effect of user level options on temporary options (see note above), if you are connected to the database as DBA, and issue:

```
SET OPTION IDENTITY_INSERT = 'Customers'
```

the value for the option is set to Customers for the user DBA and temporary for the current connection. Other users who subsequently connect to the database as DBA find their option value for IDENTITY_INSERT is Customers also.

See also

"QUERY_PLAN option" on page 434

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'
To support the lookup of default indexes three bytes wide	Rebuild your FP Index as a 3-byte FP with an IQ UNIQUE constraint value of 65537

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

"FP_LOOKUP_SIZE option" on page 386, "INDEX_ADVISOR_MAX_ROWS option" on page 397, "MINIMIZE_STORAGE option" on page 421, and "QUERY_PLAN option" on page 434.

"sp_iqindexadvice procedure," "sp_iqindexmetadata procedure," "sp_iqrebuildindex procedure," and "sp_iqrowdensity procedure," Chapter 7, "System Procedures," in *Reference: Building Blocks, Tables, and Procedures*

"The Fast Projection (FP) default index type" in Chapter 6, "Using Sybase IQ Indexes" in *System Administration Guide: Volume 1*

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

"Using IQ UNIQUE constraint on columns," Chapter 9, "Ensuring Data Integrity," in *System Administration Guide: Volume 1*

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

“FP_LOOKUP_SIZE option” on page 386 and “INDEX_ADVISOR option” on page 395

sp_iqindexadvice procedure in Chapter 7, “System Procedures” in *Reference: Building Blocks, Tables, and Procedures*

“The Fast Projection (FP) default index type” in Chapter 6, “Using Sybase IQ Indexes” in *System Administration Guide: Volume 1*

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
ON

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 for an individual connection or for the PUBLIC group. 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	0 – 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. IQGOVERN_PRIORITY_TIME must be set PUBLIC.

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.

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

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

The following example requests a hash join:

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

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 of 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). Use the SET OPTION command to adjust the amount of heap memory used by load operations.

The Sybase IQ 15.0 load process has significantly reduced heap memory usage; the LOAD_MEMORY_MB option is primarily for fixed-width loads.

If your system runs out of virtual memory, specify a value less than 2000 and decrease the value until the load works. For insertions into wide tables, you might need to set LOAD_MEMORY_MB to a low value (100-200 MB). If you set the value too low, it may be physically impossible to load the data.

The amount of virtual memory used can become quite large if many columns, such as in a very wide table, are loaded at once. The wider the table, the more the load memory. The more users doing loads, the more heap/load memory is allocated outside IQ.

There are several courses of action you can take if you encounter the following error:

```
"All available virtual memory has been used ..."
```

You can set an upper limit on the amount of virtual memory a LOAD command can use by setting LOAD_MEMORY_MB to a non-zero value, with 2000MB the maximum allowed value.

You can also adjust BLOCK FACTOR or BLOCK SIZE LOAD command options. These command options default to 10000 and 500000, respectively, but you can set them to any number. Setting them lower forces the load to use less virtual memory.

You can also resort to loading a subset of the columns at a time, which is referred to as a partial-width load.

See also

"SET OPTION statement" on page 307

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 425

"LOAD TABLE statement" on page 230

LOCKED option

Function

If set for a login policy, prevents users with that policy from establishing new connections. For details, see *SQL Anywhere Server – Database Administration*.

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 *System Administration Guide: Volume 1*.

LOGIN_PROCEDURE option

Function

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

Allowed values

String

Default

sp_login_environment system procedure

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

Description — The initial connection compatibility options settings are controlled using the LOGIN_PROCEDURE option, which is called after all the checks have been performed to verify that the connection is valid. The LOGIN_PROCEDURE option names a stored procedure to run when users connect. The default setting is to use the sp_login_environment system stored procedure. You can specify a different stored procedure. The procedure specified by the LOGIN_PROCEDURE option is not executed for event connections.

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

For more details on the LOGIN_PROCEDURE option and examples, see "login_procedure option [database]" in *SQL Anywhere Server – Database Administration > Configuring Your Database > Database options > Introduction to database options > Alphabetical list of options*.

See also — "Initial option settings" on page 338

"sp_login_environment system procedure" and "sp_tsql_environment system procedure" in Chapter 7, "System Procedures" in *Reference: Building Blocks, Tables, and Procedures*

"Managing IQ user accounts and connections" in Chapter 8, "Managing User IDs and Permissions" in the *System Administration Guide: Volume 1*

MAIN_RESERVED_DBSPACE_MB option

Function — Controls the amount of space Sybase IQ reserves in the IQ main store.

Allowed values — Integer greater than or equal to 200, in megabytes

Default — 200; Sybase IQ actually reserves a maximum of 50% and a minimum of 1% of the last read-write file in IQ_SYSTEM_MAIN

Scope — Can be set only for the PUBLIC group. Requires DBA permissions to set the option. Takes effect immediately. The server does *not* need to be restarted in order to change reserved space size.

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.

Reserved space size is calculated as a maximum of 50% and a minimum of 1% of the last read-write file in IQ_SYSTEM_MAIN.

See also

"IQ main store and IQ temporary store space management" in Chapter 5, "Working with Database Objects" in the *System Administration Guide: Volume 1*

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

100000000

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 414

To control precision for queries on the catalog store, see "PRECISION option" on page 430

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 413

To control scale for queries on the catalog store, see "SCALE option" on page 443

MAX_CONNECTIONS option

Function

Specifies the maximum number of concurrent connections allowed for a user. For details, see *SQL Anywhere Server – Database Administration*.

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 – 4294967295

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* exceeds the default 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_DAYS_SINCE_LOGIN option

Function	Specifies the maximum number of days that can elapse between two successive logins by the same user. For details, see “Managing login policies overview” in *SQL Anywhere Server – Database Administration > Configuring Your Database > Managing user IDs, authorities, and permissions*.

MAX_FAILED_LOGIN_ATTEMPTS option

Function	Specifies the maximum number of failed attempts, since the last successful attempt, to log into the user account before the account is locked. For details, see “Managing login policies overview” in *SQL Anywhere Server – Database Administration > Configuring Your Database > Managing user IDs, authorities, and permissions*.

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 4294967295
Default	2500000
Scope	Can be set temporary for an individual connection or 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 – 10000

Default
144

Scope
Can be temporary or permanent. Does not require 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 – 10000

Default
144

Scope
Can be temporary or permanent. Does not require 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
This parameter sets an upper bound which limits how parallel the optimizer will permit query operators to go. This can influence the CPU usage for many query join, GROUP BY, UNION, ORDER BY and other query operators.

Systems with more than 24 CPU cores often benefit from a larger value, up to the total number of CPU cores on the system; you can experiment to find the best value for this parameter for your system and queries.

Systems with 24 or fewer CPU cores should not need to reduce this value, unless excessive system time is seen. In that case, you can try reducing this value to determine if that adjustment can lower the CPU system time and improve query response times and overall system throughput.

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_TEMP_SPACE_PER_CONNECTION option

Function — Limits temporary store space used per connection.

Allowed values — Integer (number of MB)

Default — 0 (no limit on temporary store usage)

Scope — DBA permissions are required to set this option. Can be set temporary for an individual connection or for the PUBLIC group. Takes effect immediately.

Description — By controlling space per connection, this option enables DBAs to manage the space for both loads and queries. If the connection exceeds the run time quota specified by MAX_TEMP_SPACE_PER_CONNECTION, Sybase IQ rolls back the current statement and returns this message to the IQ message file or client user:

```
"The current operation has been cancelled:
Max_Temp_Space_Per_Connection exceeded"
```

Conditions that may fill the buffer cache include read or write errors, lack of main or temp space, or being out of memory. Sybase IQ may return the first error encountered in these situations and the DBA must determine the appropriate solution. For more information, see *Error Messages* and Chapter 14, “Troubleshooting Hints” in *System Administration Guide: Volume 1*.

Examples — This statement sets a 500GB limit for all connections:

```
SET OPTION
PUBLIC.MAX_TEMP_SPACE_PER_CONNECTION = 512000
```

This statement sets a 10TB limit for all connections:

```
SET OPTION
PUBLIC.MAX_TEMP_SPACE_PER_CONNECTION = 10485760
```

This statement sets a 5000MB limit for user wilson:

```
SET OPTION
wilson.MAX_TEMP_SPACE_PER_CONNECTION = 5000
```

See also — “QUERY_TEMP_SPACE_LIMIT option” on page 439

MAX_WARNINGS option

Function — Controls the maximum number of warnings allowed.

Allowed values — Any integer

Default — 2^{48} - 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 not 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 the specialized storage optimization does not occur for all newly created columns; when MINIMIZE_STORAGE 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.

Note An IQ UNIQUE value greater than 65536 can allow the creation of 3-byte indexes, whereas previously such values were used to prevent it with MINIMIZE_STORAGE ON. If you want to prevent the specialized storage optimization with MINIMIZE_STORAGE ON, give IQ UNIQUE a constraint value greater than 16777216.

When the ratio of main memory to the number of columns is large, turning MINIMIZE_STORAGE ON is beneficial. Otherwise, storage of new columns generally benefits from turning this option OFF.

Specifying IQ UNIQUE explicitly in CREATE TABLE or ALTER TABLE ADD COLUMN overrides the MINIMIZE_STORAGE option for that column.

See also

"FP_LOOKUP_SIZE option" on page 386 and "INDEX_ADVISOR option" on page 395

"The Fast Projection (FP) default index type" in Chapter 6, "Using Sybase IQ Indexes" in *System Administration Guide: Volume 1*

Chapter 5, "Working with Database Objects" and Chapter 6, "Using Sybase IQ Indexes" (particularly "FP(3) index") in *System Administration Guide: Volume 1*

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
```

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'
```

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 Operations such as INSERT...VALUES, LOAD, and SYNCHRONIZE are not affected by the NOEXEC option because they do not include a query.

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.

See also

“EARLY_PREDICATE_EXECUTION option” on page 383

“GRAPHICAL_PLAN function [String]” and “HTML_PLAN function [String]” in Chapter 4, “SQL Functions” in *Reference: Building Blocks, Tables, and Procedures*

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.

ODBC_DISTINGUISH_CHAR_AND_VARCHAR option

Function
Controls how the Sybase IQ and SQL Anywhere ODBC driver describes CHAR columns.

Allowed values
ON, OFF

Default
OFF

Description
When a connection is opened, the Sybase IQ and SQL 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 3, “SQL Data Types” in *Reference: Building Blocks, Tables, and Procedures*

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_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.
- CONTINUE – 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 120

CREATE PROCEDURE statement [T-SQL] on page 127

"CONTINUE_AFTER_RAISERROR option [TSQL]" on page 361

"Transact-SQL procedure language overview" in Appendix A, "Compatibility with Other Sybase Databases" in *Reference: Building Blocks, Tables, and Procedures*

Appendix A, "Compatibility with Other Sybase Databases" in *Reference: Building Blocks, Tables, and Procedures*

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 4, "Managing System Resources" in the *Performance and Tuning Guide*

PASSWORD_EXPIRY_ON_NEXT_LOGIN option

Function

When a user is assigned a login policy and this option for the policy is set ON, then the user's password is marked for expiry immediately upon next login. For details, see "CREATE LOGIN POLICY statement" in *SQL Anywhere Server – SQL Reference* > Using SQL > SQL statements > SQL statements (A-D).

PASSWORD_GRACE_TIME option

Function

The number of days before password expiration during which login is allowed but the default post login procedure issues warnings. For details, see "CREATE LOGIN POLICY statement" in *SQL Anywhere Server – SQL Reference* > Using SQL > SQL statements > SQL statements (A-D).

PASSWORD_LIFE_TIME option

Function

The maximum number of days before a password must be changed. For details, see "CREATE LOGIN POLICY statement" in *SQL Anywhere Server – SQL Reference* > Using SQL > SQL statements > SQL statements (A-D).

POST_LOGIN_PROCEDURE option

Function

Specifies a login procedure whose result set contains messages that are displayed by the client application immediately after a user successfully logs in.

Allowed values

String

Default

dbo.sa_post_login_procedure

Scope

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

Description

The default post login procedure, dbo.sa_post_login_procedure, executes immediately after a user successfully logs in.

If you have DBA authority, you can customize the post login actions by creating a new procedure and setting POST_LOGIN_PROCEDURE to call the new procedure. *Do not* edit dbo.sa_post_login_procedure. The customized post login procedure must be created in every database you use.

The post login procedure supports the client applications dbisql, dbisqlc, and the IQ plug-in for Sybase Central.

See also

"LOGIN_PROCEDURE option" on page 411

"Managing IQ user accounts and connections" in Chapter 8, "Managing User IDs and Permissions" in the *System Administration Guide: Volume 1*

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

Scope

Only PUBLIC setting allowed.

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 443

For queries on the IQ store, see "MAX_CLIENT_NUMERIC_PRECISION option" on page 413

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	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	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
Scope	Only PUBLIC setting allowed.

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 PUBLIC group.

QUERY_DETAIL option

Function

Specifies whether or not to include additional query information in the Query Detail section of the query 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 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 or message log archiving to avoid filling up your message log file. For details, see “Message log wrapping” in Chapter 1, “Overview of Sybase IQ System Administration” of the *System Administration Guide: Volume 1*.

See also

“QUERY_PLAN option” on page 434

“QUERY_PLAN_AS_HTML option” on page 435

QUERY_NAME option

Function

Gives a name to an executed query in its query plan.

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. The query name is also added to the filename for HTML query plans. This option has no other effect on the query.

QUERY_PLAN option

Function
Specifies whether or not additional query plans are printed to the Sybase IQ message file.

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 textual query plans in the IQ message file. These query plans display the query tree topography, as well as details about optimization and execution. 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 433

"QUERY_PLAN_AS_HTML option" on page 435

"QUERY_PLAN_AFTER_RUN option" on page 435

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.
See also	“QUERY_PLAN_AS_HTML option” on page 435 “GRAPHICAL_PLAN function [String]” and “HTML_PLAN function [String]” in Chapter 4, “SQL Functions” in *Reference: Building Blocks, Tables, and Procedures*

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_query-number*.html

For example, if the user DBA sets the temporary option QUERY_NAME to 'Query_1123', a file created on May 18, 2009 at exactly 8:30 a.m. is called *DBA_Query_1123_20090518_083000_1.html*. The date, time, and unique number 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. Enable IQ message log wrapping or message log archiving to avoid filling up your message log file. For details, see "Message log wrapping" in Chapter 1, "Overview of Sybase IQ System Administration" of the *System Administration Guide: Volume 1*.

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.

See also

"QUERY_PLAN_AFTER_RUN option" on page 435

"GRAPHICAL_PLAN function [String]" and "HTML_PLAN function [String]" in Chapter 4, "SQL Functions" in *Reference: Building Blocks, Tables, and Procedures*

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 by allowing HTML query plans to be produced outside of the server directory. 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 435

QUERY_PLAN_TEXT_ACCESS option

Function

Enables or prevents users from accessing query plans from the Interactive SQL (dbisql) client or from using SQL functions to get plans.

Allowed values

ON, OFF

Default

OFF

Scope

DBA permissions are required to modify this option. Can be set temporary for an individual connection or for the PUBLIC group. Takes effect immediately.

Description

When QUERY_PLAN_TEXT_ACCESS option is ON, users can view, save, and print query plans from the dbisql client. When the option is OFF, query plans are not cached, and other query plan-related database options have no affect on the query plan display from the dbisql client. The following error message displays:

```
No plan available. The database option
QUERY_PLAN_TEXT_ACCESS is OFF.
```

See also

"QUERY_DETAIL option" on page 433

"QUERY_PLAN_AS_HTML option" on page 435

"QUERY_PLAN_AFTER_RUN option" on page 435

"QUERY_PLAN_TEXT_CACHING option" on page 438

"GRAPHICAL_PLAN function [String]" and "HTML_PLAN function [String]" in Chapter 4, "SQL Functions" in *Reference: Building Blocks, Tables, and Procedures*

"OUTPUT statement [DBISQL]" on page 263

"WRITE_CLIENT_FILE function [String]" and "PLAN function [Miscellaneous]" in *SQL Anywhere Server – SQL Reference > Using SQL > SQL functions > SQL functions (P-Z)*

"EXPLANATION function [Miscellaneous]" and "GRAPHICAL_PLAN function [Miscellaneous]" in *SQL Anywhere Server – SQL Reference > Using SQL > SQL functions > SQL functions (E-O)*

QUERY_PLAN_TEXT_CACHING option

Function

Allow users to specify whether or not Sybase IQ generates and caches IQ plans for queries executed by the user.

Allowed values

ON, OFF

Default

OFF

Scope

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

Description

IQ query plans vary in size and can become very large for complex queries. Caching plans for display on the dbisql client can have high resource requirements. The QUERY_PLAN_TEXT_CACHING option gives users a mechanism to control resources for caching plans. With this option turned OFF (the default), the query plan is not cached for that user connection.

Note If QUERY_PLAN_TEXT_ACCESS is turned OFF, the query plan is not cached for the connections from that user, no matter how QUERY_PLAN_TEXT_CACHING is set.

See also

"QUERY_DETAIL option" on page 433

"QUERY_PLAN_TEXT_ACCESS option" on page 437

"QUERY_PLAN_AS_HTML option" on page 435

"QUERY_PLAN_AFTER_RUN option" on page 435

"GRAPHICAL_PLAN function [String]" and "HTML_PLAN function [String]" in Chapter 4, "SQL Functions" in *Reference: Building Blocks, Tables, and Procedures*

"OUTPUT statement [DBISQL]" on page 263

"PLAN function [Miscellaneous]" in *SQL Anywhere Server – SQL Reference > Using SQL > SQL functions > SQL functions (P-Z)*

"EXPLANATION function [Miscellaneous]" in *SQL Anywhere Server – SQL Reference > Using SQL > SQL functions > SQL functions (E-O)*

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

Specifies the maximum estimated amount of temp space before a query is rejected.

Allowed values Any integer

Default 0 (no limit)

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 is estimated to require 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
```

When set to zero (the default), there is no limit on temporary store usage by queries.

Users may override this option in their own environments to run queries that can potentially fill up the entire temporary store. To prevent runaway queries from filling up the temporary store, the DBA can set the option MAX_TEMP_SPACE_PER_CONNECTION. The MAX_TEMP_SPACE_PER_CONNECTION option monitors and limits actual temporary store usage for all DML statements, not just queries.

See also "MAX_TEMP_SPACE_PER_CONNECTION option" on page 420

QUERY_TIMING option

Function Determines whether or not to collect specific timing statistics and display them in the query 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 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.

Query timing is represented in the query plan detail as a series of timestamps. These timestamps correspond to query operator phases (Conditions, Prepare, Fetch, Complete). HTML and dbisql query plans display query timing graphically as a timeline.

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 temporarily sets QUOTED_IDENTIFIER to ON.
See also	Appendix A, “Compatibility with Other Sybase Databases” in *Reference: Building Blocks, Tables, and Procedures*

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 *System Administration Guide: Volume 1*

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 371 "TIME_FORMAT option" on page 466 "TIMESTAMP_FORMAT option" on page 466

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. 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. If both TOP and ROW_COUNT are set, then the value of TOP takes precedence.

The ROW_COUNT option could produce non-deterministic results when used in a query involving global variables, system functions or proxy tables. Such queries are partly executed using CIS (Component Integrated Services). In such cases, use SELECT TOP *n* instead of setting ROW_COUNT, or set the global variable to a local one and use that local variable in the query.

See also

"QUERY_ROWS_RETURNED_LIMIT option" on page 439

SELECT statement on page 291

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

Scope

Can be set only for PUBLIC.

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 430

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_COLLATION option

Function

Allows implicit use of the SORTKEY function on ORDER BY expressions.

Allowed values

Internal, collation_name, or collation_id

Default

Internal

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 value of this option is Internal, the ORDER BY clause remains unchanged.

When the value of this option is set to a valid collation name or collation ID, any string expression in the ORDER BY clause is treated as if the SORTKEY function has been invoked.

Example

Set the sort collation to binary:

```
SET TEMPORARY OPTION sort_collation='binary';
```

Setting the sort collation to binary transforms the following queries:

```
SELECT Name, ID
FROM Products
ORDER BY Name, ID;
SELECT Name, ID
FROM Products
ORDER BY 1, 2;
```

The queries are transformed into:

```
SELECT Name, ID
FROM Products
ORDER BY SORTKEY(Name, 'binary'), ID;
```

See also

"SORTKEY function [String]" in Chapter 4, "SQL Functions" of *Reference: Building Blocks, Tables, and Procedures*

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 the sorts and hashes of other users 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

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

ON

Description

If the truncated characters consist only of spaces, no exception is raised. ON corresponds to SQL92 behavior. When STRING_RTRUNCATION 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.

This option was OFF by default prior to Sybase IQ 15.0. It can safely be set to OFF for backward compatibility. However, the ON setting is preferable to identify statements where truncation may cause data loss.

SUBQUERY_CACHING_PREFERENCE option

Function

Controls which algorithm to use for processing correlated subquery predicates.

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

For correlated subquery predicates, the IQ optimizer offers a choice of caching outer references and subquery results that reduces subquery execution costs. SUBQUERY_CACHING_PREFERENCE lets you override the optimizer's costing decision when choosing which algorithm to use. It does not override internal rules that determine whether an algorithm is legal within the query engine.

A setting of a non-zero value affects every subquery predicate in the query. A non-zero value cannot be used selectively for one subquery predicate in a query.

SUBQUERY_CACHING_PREFERENCE is normally used for internal testing by experienced DBAs only. It does not apply to IN subqueries. See "IN_SUBQUERY_PREFERENCE option" on page 400.

Table 2-17 illustrates the valid values for this option and their actions.

Table 2-17: SUBQUERY_CACHING_PREFERENCE values

Value	Action
1	Use sort-based processing for the first subquery predicate. Other subquery predicates that do not have the same ordering key are processed using a hash table to cache subquery results.
2	Use the hash table to cache results for all subquery predicates when it is legal. If available temp cache cannot accommodate all of the subquery results, performance may be poor.
3	Cache one previous subquery result. Does not use SORT and HASH.
-1	Avoid using SORT. The IQ optimizer chooses HASH if it is legal.
-2	Avoid using HASH. The IQ optimizer chooses SORT or cache-one value if it is legal.
-3	Avoid using cache-one value. The IQ optimizer chooses either HASH or SORT if it is legal.

See also

"Disjunction of subquery predicates" in Chapter 2, "SQL Language Elements" in *Reference: Building Blocks, Tables, and Procedures*

SUBQUERY_FLATTENING_PERCENT option

Function

Allows the user to change the threshold at which the optimizer decides to transform scalar subqueries into joins.

Allowed values

0: Let optimizer cost model decide
1 - (2^{32} -1): Values greater than 0 set the percentage of references at which to flatten

Default

100

Scope

This option only applies to correlated scalar subqueries. DBA permissions are not required to set SUBUERY_FLATTENING_PERCENT. This option can be set by any user, at any level and takes effect immediately. If you set SUBUERY_FLATTENING_PERCENT to a non-default value, every scalar subquery predicate in the query is affected; this option cannot be used selectively for one scalar subquery predicate in a query.

Description

The Sybase IQ query optimizer can convert a correlated scalar subquery into an equivalent join operation to improve query performance. The SUBQUERY_FLATTENING_PERCENT option allows the user to adjust the threshold at which this optimization occurs.

SCALAR_FLATTENING_PERCENT represents a percent of estimated inner distinct values to estimated outer distinct values in a scalar subquery. As the estimated percent approaches 100%, the cost of evaluating the subquery as a join is likely to be smaller than using individual index probes. The value may be set larger than 100%, since the estimated inners are not guaranteed to be less than estimated outers.

See also

"SUBQUERY_FLATTENING_PREFERENCE option" on page 449

SUBQUERY_FLATTENING_PREFERENCE option

Function

Allows a user to override the decisions of the optimizer when transforming (flattening) scalar or EXISTS subqueries into joins.

Allowed values

-3 to 3

-3: Avoid flattening both EXISTS and scalar subqueries to a join operation
-2: Avoid flattening a scalar subquery to a join operation
-1: Avoid flattening an EXISTS subquery to a join operation
0: Allow the IQ optimizer to decide to flatten subqueries
1: Ignore cost flattening EXIST, if possible
2: Ignore cost flattening scalar, if possible
3: Ignore cost of both EXISTS and scalar subquery

Default

0

Scope

DBA permissions are not required to set this option. SUBQUERY_FLATTENING_PREFERENCE can be set by any user at any level. This option takes effect immediately. If you set the option to a non-zero value, every subquery predicate in the query is affected; this option can not be used selectively for one subquery predicate in a query.

Description

The Sybase IQ optimizer may convert a correlated scalar subquery or an EXISTS or NOT EXISTS subquery into an equivalent join operation to improve query performance. This optimization is called **subquery flattening**. The SUBQUERY_FLATTENING_PREFERENCE option allows you to override the costing decision of the optimizer when choosing the algorithm to use.

Setting SUBQUERY_FLATTENING_PREFERENCE to 0 (allow the IQ optimizer to decide to flatten subqueries) is equivalent to setting the now deprecated FLATTEN_SUBQUERIES option to ON in earlier releases of Sybase IQ.

See also

"SUBQUERY_FLATTENING_PERCENT option" on page 448

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 2-18 describes the valid values for this option and their actions.

Table 2-18: SUBQUERY_PLACEMENT_PREFERENCE values

Value	Action
0	Let the optimizer choose.
1	Prefer the highest possible location in the query plan, thereby delaying the execution of the subquery to as late as possible within the query.
-1	Prefer the lowest possible location 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 471

Chapter 5, "Monitoring and Tuning Performance" in the *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_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 *System Administration Guide: Volume 1*.

“TEMP_EXTRACT_NAMEn options” on page 456

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 2-19: 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 Chapter 7, "Moving Data In and Out of Databases" in the *System Administration Guide: Volume 1*.

"TEMP_EXTRACT_SWAP option" on page 463

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 459

"TEMP_EXTRACT_QUOTES option" on page 460

"TEMP_EXTRACT_ROW_DELIMITER option" on page 461

"TEMP_EXTRACT_QUOTES_ALL option" on page 461

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 *System Administration Guide: Volume 1*.

TEMP_EXTRACT_DIRECTORY option

Function

Controls whether a user is allowed to use the data extraction facility. Also controls the directory into which temp extract files are placed and overrides a directory path specified in the TEMP_EXTRACT_NAMEn options.

Allowed values

string

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. 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: "You do not have permission to perform Extracts".

If TEMP_EXTRACT_DIRECTORY is set to FORBIDDEN for the PUBLIC group, then no one can run data extraction.

If TEMP_EXTRACT_DIRECTORY is set to a valid directory path, temp extract files are placed in that directory, overriding a path specified in the TEMP_EXTRACT_NAMEn options.

If TEMP_EXTRACT_DIRECTORY is set to an invalid directory path, an error occurs: “Files does not exist File: <invalid path>”

If TEMP_EXTRACT_DIRECTORY is blank, then temp extract files are placed in directories according to their specification in TEMP_EXTRACT_NAMEn. If no path is specified as part of TEMP_EXTRACT_NAMEn, the extract files are by default placed in the server startup directory.

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 456

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 *System Administration Guide: Volume 1*.

TEMP_EXTRACT_ESCAPE_QUOTES option

Function

Specifies whether all quotes in fields containing quotes are escaped 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 is ignored unless TEMP_EXTRACT_QUOTE is the default or set to the value of "" (double quotes), and TEMP_EXTRACT_BINARY is OFF, and either TEMP_EXTRACT_QUOTES or TEMP_EXTRACT_QUOTES_ALL is ON.

See also

“TEMP_EXTRACT_BINARY option” on page 452

“TEMP_EXTRACT_QUOTES option” on page 460

“TEMP_EXTRACT_QUOTES_ALL option” on page 461

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."

The directory path specified using the TEMP_EXTRACT_NAMEn options can be overridden with the TEMP_EXTRACT_DIRECTORY option.

See also

"TEMP_EXTRACT_DIRECTORY option" on page 454

"TEMP_EXTRACT_SIZEn options" on page 462

"TEMP_EXTRACT_APPEND option" on page 452

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 *System Administration Guide: Volume 1*.

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 *System Administration Guide: Volume 1*.

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 *System Administration Guide: Volume 1*.

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 *System Administration Guide: Volume 1*.

“TEMP_EXTRACT_COLUMN_DELIMITER option” on page 453

“TEMP_EXTRACT_QUOTES option” on page 460

“TEMP_EXTRACT_QUOTES_ALL option” on page 461

“TEMP_EXTRACT_ROW_DELIMITER option” on page 461

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 *System Administration Guide: Volume 1*.

“TEMP_EXTRACT_COLUMN_DELIMITER option” on page 453

“TEMP_EXTRACT_QUOTES option” on page 460

“TEMP_EXTRACT_QUOTES_ALL option” on page 461

“TEMP_EXTRACT_ROW_DELIMITER option” on page 461

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 *System Administration Guide: Volume 1*.

"TEMP_EXTRACT_COLUMN_DELIMITER option" on page 453

"TEMP_EXTRACT_QUOTES option" on page 460

"TEMP_EXTRACT_QUOTES_ALL option" on page 461

"TEMP_EXTRACT_ROW_DELIMITER option" on page 461

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 *System Administration Guide: Volume 1*.

“TEMP_EXTRACT_COLUMN_DELIMITER option” on page 453

“TEMP_EXTRACT_QUOTES option” on page 460

“TEMP_EXTRACT_QUOTES_ALL option” on page 461

“TEMP_EXTRACT_ROW_DELIMITER option” on page 461

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 *System Administration Guide: Volume 1*.

“TEMP_EXTRACT_NAMEn options” on page 456

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 2-20: 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 *System Administration Guide: Volume 1*.

“TEMP_EXTRACT_BINARY option” on page 452

TEMP_RESERVED_DBSPACE_MB option

Function

Controls the amount of space Sybase IQ reserves in the temporary IQ store.

Allowed values

Integer greater than or equal to 200 in megabytes

Default

200; Sybase IQ actually reserves a maximum of 50% and a minimum of 1% of the last read-write file in IQ_SYSTEM_TEMP

Scope

Can be set only for the PUBLIC group. DBA authority is required to set the option. Takes effect immediately. The server does *not* need to be restarted in order to change reserved space size.

Description

TEMP_RESERVED_DBSPACE_MB lets you control the amount of space Sybase IQ sets aside in your temporary 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.

Reserved space size is calculated as a maximum of 50% and a minimum of 1% of the last read-write file in IQ_SYSTEM_TEMP.

See also

“IQ main store and IQ temporary store space management” in Chapter 5, “Working with Database Objects” in the *System Administration Guide: Volume 1*

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

ON

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 439 and "MAX_TEMP_SPACE_PER_CONNECTION option" on page 420.

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 "sa_disk_free_space system procedure" in *SQL Anywhere Server – SQL Reference > System Objects > System procedures > Alphabetical list of system procedures*.

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 also 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 371

"RETURN_DATE_TIME_AS_STRING option" on page 442

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 2-21: 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 371

"RETURN_DATE_TIME_AS_STRING option" on page 442

TOP_NSORT_CUTOFF_PAGES option

Function
Sets the result size threshold for TOP N algorithm selection.

Allowed values
1 – 1000

Default
1

Description
The TOP_NSORT_CUTOFF_PAGES option sets the threshold, measured in pages, where evaluation of a query that contains both a TOP clause and ORDER BY clause switches algorithms from ordered list-based processing to sort-based processing. Ordered list processing performs better in cases where the TOP N value is smaller than the number of result rows. Sort-based processing performs better for large TOP N values.

In some cases, increasing TOP_NSORT_CUTOFF_PAGES can improve performance by avoiding sort-based processing.

See also
"SELECT statement" on page 291

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 362

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 for an individual connection 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';
```

For an example that defines a table and a function and sets some login policy options, see “verify_password_function option [database]” in *SQL Anywhere Server – Database Administration > Configuring Your Database > Database options > Introduction to database options > Alphabetical list of options*.

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 5, “Monitoring and Tuning Performance” in the *Performance and Tuning Guide* “SWEEPER_THREADS_PERCENT option” on page 451

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.

WD_DELETE_METHOD option

Function — Specifies the algorithm used during a delete in a WD 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 during a delete operation in a WD index. When this option is not set or is set to 0, the delete method is selected by the cost model. 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

Allowed values for WD_DELETE_METHOD:

- 0: The delete method is selected by the cost model. Cost model only selects either mid or large method for deletion.
- 1: Forces small method for deletion. Small method is useful when the number of rows being deleted is a very small percentage of the total number of rows in the table. Small delete can randomly access the index, causing cache thrashing with large datasets.

- 2: Forces large method for deletion. This algorithm scans the entire index searching for rows to delete. Large method is useful when the number of rows being deleted is a high percentage of the total number of rows in the table.
- 3: Forces mid method for deletion. Mid method is a variation of the small method that accesses the index in order and is generally faster than the small method.

Example

The following statement forces the large method for deletion from a WD index:

```
SET TEMPORARY OPTION WD_DELETE_METHOD = 2
```

See also

For more details about these delete methods, see “Optimizing delete operations” in Chapter 3, “Optimizing Queries and Deletions” in the *Performance and Tuning Guide*.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

P

Q

R

S

T

U

V

W