Reference: Building Blocks, Tables, and Procedures

SAP Sybase IQ 16.0

DOCUMENT ID: DC38151-01-1600-01
LAST REVISED: February 2013

Contents

Audience

This book is intended for SAP® Sybase® IQ users who require reference material for SQL statements, language elements, data types, functions, system procedures, and system tables.

Other books provide more context on how to perform particular tasks. Use this book to get information about SQL syntax, parameters, and options. For command line utility start-up parameters, see the *Utility Guide*.

File Locations and Installation Settings

These topics describe the installation and operating system settings used by SAP Sybase IQ.

Depending on the operating system, these settings may be stored as environment variables, initialization file entries, or registry entries.

Installation Directory Structure

When you install SAP Sybase IQ, several directories may be created. This section describes the directory structure.

The directories created depend on which options you choose during installation and which directories already exist in your Sybase directory (the directory defined by *$SYBASE* on UNIX or *%SYBASE%* on Windows).

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

By default, the SAP Sybase IQ directory is `IQ-16_0`. The location of `IQ-16_0` varies, depending on where you install SAP Sybase IQ. The `IQ-16_0` directory is also referred to by the environment variable $IQDIR16 on UNIX or %IQDIR16% on Windows.

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

- **Demo directory (`%ALLUSERSPROFILE%/SybaseIQ/demo`) –** Holds the tools required to build the iqdemo database. The iqdemo database files are `iqdemo.db`, `iqdemo.iq`, `iqdemo.iqmsg`, and `iqdemo.iqtmp`. The demo database itself is not shipped with SAP Sybase IQ.
- **The subdirectory `/demo/adata` –** Holds 16.x data to allow the creation of the 16.x iqdemo database. The subdirectory `/demo/demodata` holds SAP Sybase IQ 12.7 data to allow the creation of an iqdemo database that has the same schema layout and data as the IQ 12.7 asiqdemo database. Use `/demo/mkiqdemo.bat` on Windows and `demo/mkiqdemo.sh` on UNIX to create the 16.x iqdemo database. The iqdemo database can be used to demonstrate problems to Technical Support.
- **Scripts directory (`IQ-16_0/scripts`) –** Holds some scripts used in examples and when creating catalog objects like stored procedures. Do not edit these scripts. If you edit, delete, or move these scripts, the server will not operate properly.
- **Samples directories –** The `samples` directory contains SQL samples and user-defined function (UDF) samples. `%ALLUSERSPROFILE%/SybaseIQ/samples/`

`sqlanywhere` contains directories of SQL samples. The `sqlanywhere/c directory` holds C++ examples that illustrate using ESQL (embedded SQL) and C with SAP® Sybase SQL Anywhere. Because SQL Anywhere and SAP Sybase IQ share common code, you can modify these examples for use with SAP Sybase IQ. The `%ALLUSERSPROFILES%/SybaseIQ/samples/udf` directory holds sample C++ scalar and aggregate UDFs.

- **Executable directories –** Hold executables, libraries, help files, and so on. On UNIX, executable subdirectories include `IQ-16_0` subdirectories `/bin64`, `/lib64`, `/logfiles`, `/res`, and `/tix`. On Windows, these include `IQ-16_0` subdirectories `\h`, `\install`, `\java`, and `\bin32`.

How SAP Sybase IQ Locates Files

To start and run, SAP Sybase IQ must find and access several types of files. Several directories or files with identical names may reside on a system.

Understanding how SAP Sybase IQ finds these files is important to ensure that you use the correct files. The types of files include but are not limited to:

- Libraries – might include product libraries or system libraries. File name extensions include `.so.nnn` or `.so` on UNIX, or `.dll` or `.lib` on Windows. These files are required to run SAP Sybase IQ. If an incorrect DLL is found, a version mismatch error may occur. For example, library files might be found in `$IQDIR16/lib64` or `$SYBASE/$SYBASE_OCS/lib64` on UNIX, or `%IQDIR16%\bin32` or `%SYBASE\%SYBASE_OCS\dll` on Windows. An empty directory, `$IQDIR16/usrlib`, lets you supersede default libraries with custom libraries and patches, because **start_iq** includes `usrlib` before regular library directories. SAP Sybase IQ uses both SAP® Sybase Adaptive Server Enterprise and SQL Anywhere libraries. If either of these products have already been installed on your system, note the directory where they are installed to avoid confusion.
- Interface files – required to run SAP Sybase IQ. For example, `.odbc.ini` and `utility_db.ini` on UNIX, and `util_db.ini` on Windows. For more information about these files, see Administration: User Management and Security.
- Configuration files – used to specify connection parameters. Examples include `default.cfg` on Windows or `iqdemo.cfg`.
- Database files – store the data and metadata. For example: `iqdemo.db`, `iqdemo.iq`, `iqdemo.iqmsg`, `iqdemo.iqtmp`.
- Log files – store information about the current session on the server and connected database. For example, a server log might be named `%ALLUSERSPROFILE%/logfiles/yourservername.0006.srvlog`. The database log (for example, `%ALLUSERSPROFILE%/demo/iqdemo.log`) is created when you connect to the database. For more information about these files, see Administration: Database and Utility Guide.

Product scripts – are sample files that show how to create, populate, and upgrade databases.

- User files – include flat files used with the LOAD command and SQL scripts used with tools such as Interactive SQL.
- Temporary files – created by to store temporary information for operations like performing sorts for queries.

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

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

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

Simple File Searching

In many SQL statements such as **LOAD TABLE** or **CREATE DATABASE**, the file name is interpreted as relative to the current working directory of the database server; that is, where the server was started.

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

Extensive File Searching

SAP Sybase IQ programs, including the database server and administration utilities, carry out extensive searches for required files, such as DLLs or shared libraries.

In these cases, SAP Sybase IQ programs look for files in the following order:

- **The executable directory –** the directory in which the program executable is held. Also, directories with the following paths relative to the program executable directory:
  - Parent of the executable directory.
  - A child of the parent directory named `scripts`.
- **Current working directory –** when a program is started, it has a current working directory (the directory from which it is started). This directory is searched for required files.
- **Location registry entry –** on a Windows installation, SAP Sybase IQ adds a LOCATION registry entry. The indicated directory is searched, followed by the following:
  - A child named `scripts`
  - A child with the operating system name (`bin32`, `bin`, and so on)

- **System-specific directories –** this includes directories where common operating system files are held, such as the Windows directory and the Windows\system directory on Windows.
- **CLASSPATH directories –** for Java files, directories listed in the CLASSPATH environment variable are searched to locate files.
- **PATH directories –** directories in the system path and the user's path are searched to locate files.
- **LIBRARY PATH directories –** directories listed in the `LIBPATH` environment variable are searched for shared libraries.

Environment Variables

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

Setting Environment Variables on Windows

On Windows platforms, the installation program automatically sets all environmental variables, so no changes are necessary. However, if you must set optional variables or change defaults, use this procedure.

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

   The Environment Variables dialog opens.

   a) If the environment variable does not already exist, click New and type the variable name and its value in the spaces provided; then click OK.
   b) If the variable does exist, select it from the list of System Variables or User Variables, click Edit, and make any modifications in the Variable Value field. Then click OK to capture the setting.

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

Running UNIX Environment Source Files

Environment source files set the required environment variables on UNIX.

Issue the following command to set all required environment variables.

1. For the Bourne/Korn shell:

   ```
   . $SYBASE/IQ-16_0/IQ-16_0.sh
   ```
2. For the C shell:

```
source $SYBASE/IQ-16_0/IQ-16_0.csh;
rehash
```

Setting Environment Variables on UNIX

On UNIX platforms, running environment source files sets the required environment variables. However, if you must set optional variables or change defaults, use this procedure.

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

   ```
   echo $variable-name
   ```

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

   ```
   % echo $SYBASE
   ```

   ```
   /server1/users/test/sybase
   ```

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

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

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

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

   ```
   setenv VARIABLE "value"
   ```

IQCHARSET Environment Variable

IQCHARSET sets the default character set.

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

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

- IQCHARSET environment variable
- Query the operating system

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

Setting

```
IQCHARSET=charset
```

IQDIR16 Environment Variable

IQDIR16 identifies the location of the SAP Sybase IQ directory and is the location for other directories and files under that directory.

- `$IQDIR16/bin[64]/util_db.ini` holds the login ID and password for the utility database, `utility_db`. The installation program lets you change these from their default values, login ID "DBA" and password "sql."

- `$IQDIR16/logfiles` is the default location for the server log and backup/restore log (the backup history file). You can override this default by setting the IQLOGDIR16 environment variable.
- `$IQDIR16/demo` is the location for the `iqdemo` database files.

Setting

```
IQDIR16 = ${SYBASE}/IQ-16_0
```

Operating System

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

IQLANG Environment Variable

IQLANG sets the default language.

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

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

- IQLANG environment variable
- Registry (Windows only) as set by the installer
- Query the operating system

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

Setting

```
IQLANG=language_code
```

Operating System

Optional but recommended in non-English environments.

List of Language Label Values

Valid language label values and their equivalent ISO 639 language codes. Set the two-letter ISO_639 language code in the IQ_LANG environment variable.

Language	ISO_639 Language Code	Language Label	Alternative Label
Arabic	AR	arabic	N/A
Czech	CS	czech	N/A
Danish	DA	danish	N/A
Dutch	NL	dutch	N/A

Language	ISO_639 Language Code	Language Label	Alternative Label
English	EN	us_english	english
Finnish	FI	finnish	N/A
French	FR	french	N/A
German	DE	german	N/A
Greek	EL	greek	N/A
Hebrew	HE	hebrew	N/A
Hungarian	HU	hungarian	N/A
Italian	IT	italian	N/A
Japanese	JA	japanese	N/A
Korean	KO	korean	N/A
Lithuanian	LT	lithuanian	N/A
Norwegian	NO	norwegian	norweg
Polish	PL	polish	N/A
Portuguese	PT	portuguese	portugue
Russian	RU	russian	N/A
Simplified Chinese	ZH	chinese	simpchin
Spanish	ES	spanish	N/A
Swedish	SV	swedish	N/A
Thai	TH	thai	N/A
Traditional Chinese	TW	tchinese	tradchin
Turkish	TR	turkish	N/A
Ukrainian	UK	ukrainian	N/A

IQLOGDIR16 Environment Variable

The IQLOGDIR16 environment variable defines the location of various log files. IQLOGDIR16 is not set by the installation program.

- The server log is in the file `servername.nnnn.srvlog` (where nnnn is the number of times the server has been started) in the directory specified by $IQLOGDIR16.

If IQLOGDIR16 is not set to a valid, write-enabled directory, then most utilities, including **start_iq**, use the default location `$IQDIR16/logfiles` for all server logs.

Setting

```
IQLOGDIR16 = path
```

Operating System
Optional.

IQTMP16 Environment Variable

The IQTMP16 environment variable is not set by the installation program. IQTMP16 is used by SAP Sybase IQ to indicate a directory where temporary files are kept.

The IQTMP16 environment variable should point to a local directory for those using NFS (network file system), which permits the IQTMP16 directory to purge directories and files that are no longer needed as client connections are closed. Each client connection creates several directories and files in a temporary directory. These are needed only for the duration of the connection. The directory must have write permissions for all users who connect to the server.

Note: The temporary files whose location is defined by IQTMP16 are files used by the client and server. This variable does not control the default location of your IQ temporary store. The **CREATE DATABASE** statement controls the default location of your IQ temporary store.

Warning! Do not set IQTMP16 to $SYBASE or $IQDIR16.

If you do not explicitly set IQTMP16, IQTMP16 is set to a subdirectory in the UNIX directory `/tmp`.

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

To avoid conflicts when using shared memory:

- Create a temporary directory dedicated to each server. Make sure that each local client uses the same temporary directory as its server by setting the IQTMP16 environment variable explicitly in both environments.
- Create a data source name in the `.odbc.ini` file (on UNIX) for each server and provide detailed connection information.
- Use connection strings that specify explicit parameters instead of relying on defaults.
- Confirm connections by issuing:

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

Setting

```
IQTMP16 = temp_directory
```

Operating System

Optional on UNIX. Not used on Windows platforms.

JAVA_HOME Environment Variable

Defines the JRE home which points to directory containing bin/java.

Used if the location of the Java VM is not set in the $SYBASE_JRE6_32, $SYBASE_JRE6_64, or $SYBASE_JRE5_64 environment variables.

`JAVA_HOME` is commonly created when installing a VM.

On UNIX, run the `SYBASE.csh` (C shell) or `SYBASE.sh` (Bourne or Korn shell) environment source file to find and start the correct JRE for the IQ engine. The Java VM location specified in `JAVA_HOME` takes precedence over the location returned by `SYBASE.csh` or `SYBASE.sh`. If neither `JAVA_HOME` nor the `SYBASE.csh` or `SYBASE.sh` scripts locate the Java VM, IQ will not load a Java VM.

Settings

```
JAVA_HOME = Sybase/shared/JRE<version>
```

Operating System

Required.

LIBPATH or LD_LIBRARY_PATH Environment Variable

LIBPATH or LD_LIBRARY_PATH specifies the directories where SAP Sybase IQ shared libraries are located.

On UNIX, set the library path variable by running the environment source file.

Settings

```
For AIX:
LIBPATH = installation_path/lib
```

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

Operating System

Required. Variable name is platform dependent. UNIX only.

PATH Environment Variable

PATH is an operating system required variable that includes the directories where SAP Sybase IQ executables are located.

On Windows, the installation program modifies PATH. On UNIX, run the environment source file to include the necessary directories.

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

Setting

```
PATH = installation_path
```

Operating System

Required.

SQLCONNECT Environment Variable

SQLCONNECT specifies connection parameters that are used by several of the database administration utilities, such as Interactive SQL, **dbinfo**, and **dbstop**, when connecting to a database server.

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

This string is a list of parameter settings, of the form **parameter**=*value*, delimited by semicolons.

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

Note: Specifying connection parameters in SQLCONNECT rather than on the command line offers greater security on UNIX systems. It prevents users from being able to display your password with **ps -ef**. This is especially useful if you run Interactive SQL or other utilities in quiet mode. Note that specifying connection parameters in SQLCONNECT rather than on the command line is more secure, but is not entirely secure. Because the password is in plain text, it possible for hostile action to extract it from the environment context. For more information, see *Connection and Communication Parameters Reference* in Administration: Database.

Settings

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

Operating System

Optional.

SYBASE Environment Variable

SYBASE identifies the location of Sybase applications, such as Open Client and Open Server.

You must set the SYBASE variable before you can install SAP Sybase IQ on UNIX systems.

Setting

```
SYBASE = path
```

Operating System
Required.

$SYBASE_JRE7_64, $SYBASE_JRE7_32 Environment Variables

This variable specifies the location of the Java Runtime Environment used by Sybase Control Center.

On startup, Sybase Control Center checks SCC_JAVA_HOME for Java version definition. If SCC_JAVA_HOME is undefined, Sybase Control Center checks for installed JREs in this order:

- SYBASE_JRE7_64
- SYBASE_JRE7
- SYBASE_JRE7_32

Sybase Control Center then sets SCC_JAVA_HOME to the first value it finds in this list.

Setting
Source the `IQ.sh` (Bourne/Korn shell) `IQ.csh` (C shell) files.

Tip: Alternately, you can set the JRE manually:

```
SCC_JAVA_HOME=${SYBASE}/shared/JRE-7_(minor_version)_64BIT
```

or:

```
SCC_JAVA_HOME=${SYBASE}/shared/JRE-7_(minor_version)_32BIT
```

SYBASE_OCS Environment Variable

SYBASE_OCS specifies the home directory for the Open Client product.

This variable is only used on Windows. On Windows, the installation program sets SYBASE_OCS when it installs Open Client/Server Software Developers Kit.

Setting

```
SYBASE_OCS = "OCS-16_0"
```

Operating System
Required.

Registry Entries

On Windows operating systems, SAP Sybase IQ uses several Registry settings.

These settings are made for you by the software, and in general operation, you should not need to access the registry. The settings are provided here if you modify your operating environment.

Warning! Modifying the Registry is not recommended, as incorrect changes might damage your system.

Current User and Local Machine Settings

Some operating systems, such as Windows, hold two levels of system settings: user settings and local machine settings.

Current user settings are specific to an individual user and are used only when that user is logged on. Local machine settings are global to the machine and are available no matter which user is logged on;. You must have administrator permissions on your machine to make local machine settings.

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

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

If you make settings in both current user and local machine registries, the current user setting takes precedence over the local machine setting.

If you are running a SAP Sybase IQ program as a service on Windows, ensure that the settings are made at the local machine level.

Services can continue to run under a special account when you log off a machine, as long as you do not shut the machine down entirely. Services can be made independent of individual accounts and need access to local machine settings.

In general, use local machine settings.

Registry Structure

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

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

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

```
SOFTWARE > SAP Sybase IQ 16.0
```

Starting the Registry Editor

Start the registry editor to access the Windows registry.

1. Select **Start** > **Run**.
2. In the Open box, type:

   ```
   regedt32
   ```

Registry Settings on Installation

The installation program automatically makes these registry settings in the Sybase registry.

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

  ```
  Location:REG_SZ:C:\Program Files\Sybase
  \IQ-16_0
  ```

The SAP Sybase IQ registry includes other entries for installed applications.

SQL Language Elements

These topics provide detailed descriptions of the language elements and conventions of SAP Sybase IQ SQL.

Keywords

Each SQL statement contains one or more keywords.

SQL is not case-sensitive to keywords, but throughout the SAP Sybase IQ documentation, keywords are indicated in uppercase. For example, in this statement, `SELECT` and `FROM` are keywords:

```
SELECT *
FROM Employees
```

The following statements are equivalent to the one above:

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

Reserved Words

Some keywords in SQL are also reserved words.

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

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

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

This table lists the SQL reserved words in SAP Sybase IQ. Because SQL is not case-sensitive with respect to keywords, each of the words in this table may appear in uppercase, lowercase, or any combination of the two. All strings that differ only in capitalization from these words are reserved words.

SQL reserved words

- add

- all
- alter
- and
- any
- array
- as
- asc
- attach
- backup
- begin
- between
- bigint
- binary
- bit
- bottom
- break
- by
- call
- capability
- cascade
- case
- cast
- char
- char_convert
- character
- check
- checkpoint
- close
- comment
- commit
- compressed
- conflict
- connect
- constraint
- contains
- continue
- convert
- create
- cross

- cube
- current
- current_timestamp
- current_user
- cursor
- date
- datetimeoffset
- dbspace
- deallocate
- dec
- decimal
- declare
- default
- delete
- deleting
- desc
- detach
- distinct
- do
- double
- drop
- dynamic
- else
- elseif
- encrypted
- end
- endif
- escape
- except
- exception
- exec
- execute
- existing
- exists
- externlogin
- fetch
- first
- float
- for

- force
- foreign
- forward
- from
- full
- goto
- grant
- group
- having
- holdlock
- identified
- if
- in
- index
- inner
- inout
- insensitive
- insert
- inserting
- install
- instead
- int
- integer
- integrated
- intersect
- into
- is
- isolation
- join
- json
- kerberos
- key
- lateral
- left
- like
- limit
- lock
- login
- long

- match
- membership
- merge
- message
- mode
- modify
- natural
- nchar
- new
- no
- noholdlock
- not
- notify
- null
- numeric
- nvarchar
- of
- off
- on
- open
- openstring
- openxml
- option
- options
- or
- order
- others
- out
- outer
- over
- passthrough
- precision
- prepare
- primary
- print
- privileges
- proc
- procedure
- publication

- raiserror
- readtext
- real
- reference
- references
- refresh
- release
- remote
- remove
- rename
- reorganize
- resource
- restore
- restrict
- return
- revoke
- right
- rollback
- rollup
- row
- rowtype
- save
- savepoint
- scroll
- select
- sensitive
- session
- set
- setuser
- share
- smallint
- some
- spatial
- sqlcode
- sqlstate
- start
- stop
- subtrans
- subtransaction

- synchronize
- table
- temporary
- then
- time
- timestamp
- tinyint
- to
- top
- tran
- treat
- trigger
- truncate
- tsequal
- unbounded
- union
- unique
- uniqueidentifier
- unknown
- unnest
- unsigned
- update
- updating
- user
- using
- validate
- values
- varbinary
- varbit
- varchar
- variable
- varray
- varying
- view
- wait
- waitfor
- when
- where
- while

- window
- with
- within
- work
- writetext
- xml

See also

- *Identifiers* on page 24
- *The quoted_identifier Option* on page 38
- *Subqueries in Search Conditions* on page 42
- *Column Names in Expressions* on page 29

Identifiers

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

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

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

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

Note: These characters are not allowed inside an identifier:

- **[–** square bracket (open)
- **] –** square bracket (close)
- **` –** back tick / grave accent

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

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

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

Limitations

Identifiers have the following limitations:

- Table names cannot contain double quotes.
- User names cannot contain double quote or semi-colon characters (single quote characters are allowed).
- Database names cannot contain double quote, single quote, and semi-colon characters.
- User names and database names cannot start or end with a space.
- Dbspace names are always case-insensitive, regardless of the **CREATE DATABASE**...**CASE IGNORE** or **CASE RESPECT** specification.

Database server naming restrictions exist when using the `-n` start_iq server option.

Examples

The following are all valid identifiers.

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

See also

- *Reserved Words* on page 17
- *The quoted_identifier Option* on page 38
- *Subqueries in Search Conditions* on page 42
- *Column Names in Expressions* on page 29

Database Server Naming Restrictions

If you use the **-n** switch in **start_iq [server-options]**, certain naming restrictions apply.

No character set is conversion performed on the server name. If the client character set and the database server character set differ, using extended characters in the server name can cause the server to not be found. If clients and servers run on different operating systems or locales, use 7-bit ASCII characters in the server name.

Database server names must be valid identifiers. Long database server names are truncated to different lengths depending on the protocol. Database server names cannot:

- Begin with white space, single quotes, or double quotes
- End with white space
- Contain semicolons
- Exceed 128 bytes

Note: On Windows and UNIX, SAP Sybase IQ 12.7 and earlier clients cannot connect to SAP Sybase IQ 16.0 database servers with names longer than the following lengths:

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

The server name specifies the name to be used on client application connection strings or profiles. Running multiple database servers with the same name is not recommended.

Strings

Strings are either literal strings, or expressions with CHAR or VARCHAR data types.

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

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

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

Special Characters in Strings

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

- To represent an apostrophe inside a string, use two apostrophes in a row. For example:
  ```
  'John''s database'
  ```
- To represent a newline character, use a backslash followed by n (\n). For example:
  ```
  'First line:\nSecond line:'
  ```
- To represent a backslash character, use two backslashes in a row (\\). For example:
  ```
  'c:\\temp'
  ```
- Hexadecimal escape sequences can be used for any character, printable or not. A hexadecimal escape sequence is a backslash followed by an x followed by two hexadecimal digits (for example, \x6d represents the letter m). For example:
  ```
  '\x00\x01\x02\x03'
  ```

Compatibility

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

See also

- *Comparison Conditions* on page 41
- *Expressions* on page 27
- *NULL Value* on page 75
- *Search Conditions* on page 39
- *Three-Valued Logic* on page 53

- *SQL Operators* on page 30
- *Subqueries in Search Conditions* on page 42

Expressions

Expressions are formed from several different kinds of elements, such as constants, column names, SQL operators, and subqueries.

Syntax

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

Parameters

```
case-expression:
{ CASE search-condition
... WHEN expression THEN expression [ , … ]
... [ ELSE expression ]
END
| CASE
... WHEN search-condition THEN expression [ , … ]
... [ ELSE expression ]
END }
```

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

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

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

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

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

Usage
Anywhere.

Authorization
Must be connected to the database.

Side effects
None.

Compatibility

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

See also
- *Comparison Conditions* on page 41
- *NULL Value* on page 75
- *Search Conditions* on page 39
- *Strings* on page 26
- *Three-Valued Logic* on page 53
- *SQL Operators* on page 30
- *Subqueries in Search Conditions* on page 42
- *Special Values* on page 63
- *CASE Statement Support* on page 841

Constants in Expressions

Constants are numbers or strings.

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

See also
- *Column Names in Expressions* on page 29
- *Subqueries in Expressions* on page 29
- *SQL Operators* on page 30
- *IF Expressions* on page 35

- *CASE Expressions* on page 35
- *Compatibility of Expressions and Constants* on page 37

Column Names in Expressions

A column name is an identifier preceded by an optional correlation name. A correlation name is usually a table name.

If a column name has characters other than letters, digits, and underscores, the name must be surrounded by quotation marks (“”). For example, the following are valid column names:

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

See also

- *Constants in Expressions* on page 28
- *Subqueries in Expressions* on page 29
- *SQL Operators* on page 30
- *IF Expressions* on page 35
- *CASE Expressions* on page 35
- *Compatibility of Expressions and Constants* on page 37
- *Subqueries in Search Conditions* on page 42
- *Reserved Words* on page 17
- *Identifiers* on page 24

Subqueries in Expressions

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

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

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

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

See also

- *Constants in Expressions* on page 28
- *Column Names in Expressions* on page 29

- *SQL Operators* on page 30
- *IF Expressions* on page 35
- *CASE Expressions* on page 35
- *Compatibility of Expressions and Constants* on page 37

SQL Operators

These topics describe the arithmetic, string, and bitwise operators available in SAP Sybase IQ.

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

See also

- *Constants in Expressions* on page 28
- *Column Names in Expressions* on page 29
- *Subqueries in Expressions* on page 29
- *IF Expressions* on page 35
- *CASE Expressions* on page 35
- *Compatibility of Expressions and Constants* on page 37
- *Comparison Conditions* on page 41
- *Expressions* on page 27
- *NULL Value* on page 75
- *Search Conditions* on page 39
- *Strings* on page 26
- *Three-Valued Logic* on page 53

Arithmetic Operators

These arithmetic operators are available in SAP Sybase IQ.

Table 1. Arithmetic operators

Operator	Description
expression + expression	Addition. If either expression is the NULL value, the result is the NULL value.
expression - expression	Subtraction. If either expression is the NULL value, the result is the NULL value.
- expression	Negation. If the expression is the NULL value, the result is the NULL value.

Operator	Description
expression * expression	Multiplication. If either expression is the NULL value, the result is the NULL value.
expression / expression	Division. If either expression is the NULL value or if the second expression is 0, the result is the NULL value.
expression % expression	Modulo finds the integer remainder after a division involving two whole numbers. For example, 21 % 11 = 10 because 21 divided by 11 equals 1 with a remainder of 10.

See also

- *String Operators* on page 31
- *Bitwise Operators* on page 32
- *Join Operators* on page 34
- *Operator Precedence* on page 34

String Operators

These string operators are available in SAP Sybase IQ.

Table 2. String operators

Operator	Description
expression || expression	String concatenation (two vertical bars). If either string is the NULL value, the string is treated as the empty string for concatenation.
expression + expression	Alternative string concatenation. When using the + concatenation operator, you must ensure the operands are explicitly set to character data types rather than relying on implicit data conversion.

The result data type of a string concatenation operator is a `LONG VARCHAR`. If you use string concatenation operators in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **LEFT** to the correct data type and size.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. The || operator is the ISO/ANSI SQL string concatenation operator.
- Sybase—The + operator is supported by Adaptive Server Enterprise.

See also

- *Arithmetic Operators* on page 30
- *Bitwise Operators* on page 32
- *Join Operators* on page 34
- *Operator Precedence* on page 34
- *REVERSE Function [String]* on page 324

Bitwise Operators

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

Operator	Description
&	AND
|	OR
^	EXCLUSIVE OR
~	NOT

See also

- *Arithmetic Operators* on page 30
- *String Operators* on page 31
- *Join Operators* on page 34
- *Operator Precedence* on page 34

The AND Operator (&)

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

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

See also

- *Bitwise OR (|)* on page 33
- *EXCLUSIVE OR (^)* on page 33
- *NOT (~)* on page 33

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

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

See also
- *The AND Operator (&)* on page 32
- *EXCLUSIVE OR (^)* on page 33
- *NOT (~)* on page 33

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

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

See also
- *The AND Operator (&)* on page 32
- *Bitwise OR (|)* on page 33
- *NOT (~)* on page 33

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

Bit	~ Bit
1	0
0	1

See also
- *The AND Operator (&)* on page 32

- *Bitwise OR (|)* on page 33
- *EXCLUSIVE OR (^)* on page 33

Join Operators

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

Compatibility

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

  ```
  SELECT 123 + 456
  ```

  whereas the following query returns the character string 123456:

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

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

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

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

See also

- *Arithmetic Operators* on page 30
- *String Operators* on page 31
- *Bitwise Operators* on page 32
- *Operator Precedence* on page 34

Operator Precedence

Follow this recommendation to make the order of operation explicit.

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

See also

- *Arithmetic Operators* on page 30
- *String Operators* on page 31
- *Bitwise Operators* on page 32
- *Join Operators* on page 34

IF Expressions

The IF expression provides IF-THEN-ELSE SQL expressions.

The syntax of the IF expression is as follows:

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

This expression returns:

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

Note: IF statement is different from IF expression.

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

See also

- *Constants in Expressions* on page 28
- *Column Names in Expressions* on page 29
- *Subqueries in Expressions* on page 29
- *SQL Operators* on page 30
- *CASE Expressions* on page 35
- *Compatibility of Expressions and Constants* on page 37

CASE Expressions

The **CASE** expression provides conditional SQL expressions.

You can use case expressions anywhere you can use an expression. The syntax of the **CASE** expression is as follows:

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

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

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

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

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

An alternative syntax is:

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

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

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

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

See also

- *Constants in Expressions* on page 28
- *Column Names in Expressions* on page 29
- *Subqueries in Expressions* on page 29
- *SQL Operators* on page 30
- *IF Expressions* on page 35
- *Compatibility of Expressions and Constants* on page 37
- *NULLIF Function [Miscellaneous]* on page 288
- *NULLIF Function for Abbreviated CASE Expressions* on page 36

NULLIF Function for Abbreviated CASE Expressions

The **NULLIF** function provides a way to write some **CASE** statements in short form.

The syntax for **NULLIF** is as follows:

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

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

See also

- *CASE Expressions* on page 35
- *NULLIF Function [Miscellaneous]* on page 288

Compatibility of Expressions and Constants

These topics describe the compatibility of expressions and constants between Adaptive Server Enterprise and SAP Sybase IQ.

See also

- *Constants in Expressions* on page 28
- *Column Names in Expressions* on page 29
- *Subqueries in Expressions* on page 29
- *SQL Operators* on page 30
- *IF Expressions* on page 35
- *CASE Expressions* on page 35

Compatibility of Expressions

This table describes the compatibility of expressions between Adaptive Server Enterprise and SAP Sybase IQ.

This table is a guide only, and a marking of **Both** may not mean that the expression performs in an identical manner for all purposes under all circumstances. For detailed descriptions, see the Adaptive Server Enterprise documentation and the SAP Sybase IQ documentation on the individual expression.

In this table, **expr** represents an expression, and **op** represents an operator.

Table 3. Compatibility of expressions between Adaptive Server Enterprise and SAP Sybase IQ

Expression	Supported By
constant	Both
column name	Both
variable name	Both
function (expr)	Both
- expr	Both
expr op expr	Both
(expr)	Both
(subquery)	Both

Expression	Supported By
if-expression	SAP Sybase IQ only

Compatibility of Constants

This table describes the compatibility of constants between Adaptive Server Enterprise and SAP Sybase IQ.

This table is a guide only, and a marking of *Both* may not mean that the expression performs in an identical manner for all purposes under all circumstances. For detailed descriptions, see the Adaptive Server Enterprise documentation and the SAP Sybase IQ documentation on the individual expression.

Table 4. Compatibility of constants between Adaptive Server Enterprise and SAP Sybase IQ

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

Default Interpretation of Delimited Strings

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

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

The quoted_identifier Option

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

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

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

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

```
SET quoted_identifier ON
```

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

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

```
SET quoted_identifier OFF
```

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

Examples

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

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

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

```
SELECT *
FROM Employees
WHERE Surname = "Chin"
```

See also

- *Reserved Words* on page 17
- *Identifiers* on page 24

Search Conditions

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

SQL conditions do not follow Boolean logic, where conditions are either true or false. In SQL, every condition evaluates as one of TRUE, FALSE, or UNKNOWN. This is called three-valued logic. The result of a comparison is UNKNOWN if either value being compared is the NULL value.

Rows satisfy a search condition if and only if the result of the condition is TRUE. Rows for which the condition is UNKNOWN do not satisfy the search condition.

Subqueries form an important class of expression that is used in many search conditions.

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

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

Syntax

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

Parameters

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

Usage

Anywhere

Authorization

Must be connected to the database

Example

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

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

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

Side Effects

None

See also

- *Comparison Conditions* on page 41
- *Expressions* on page 27
- *NULL Value* on page 75
- *Strings* on page 26
- *Three-Valued Logic* on page 53
- *SQL Operators* on page 30
- *Subqueries in Search Conditions* on page 42

Comparison Conditions

Comparison conditions in search conditions use a comparison operator.

The syntax for comparison conditions is as follows:

```
expression compare expression
```

where *compare* is a comparison operator. This table lists the comparison operators available in SAP Sybase IQ.

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

Example

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

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

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

Note: All string comparisons are:

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

---

Compatibility

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

See also

- *Expressions* on page 27
- *NULL Value* on page 75
- *Search Conditions* on page 39
- *Strings* on page 26
- *Three-Valued Logic* on page 53
- *SQL Operators* on page 30
- *Subqueries in Search Conditions* on page 42

Subqueries in Search Conditions

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

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

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

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

Subqueries cannot be used inside a **CONTAINS** or **LIKE** predicate.

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

```
CREATE VARIABLE @ln int;SELECT @ln = 1;select count(*) FROM
lineitemWHERE l_shipdate IN (select l_shipdate FROM lineitem WHERE
l_orderkey IN (2,4,6))OR l_shipdate IN (select l_shipdate FROM
lineitem WHERE l_orderkey IN (1,3,5))OR l_linenumber = @ln;
```

Similar subqueries joined by **AND** and **BETWEEN** are allowed.

See also

- *Column Names in Expressions* on page 29
- *Reserved Words* on page 17
- *Identifiers* on page 24
- *Comparison Conditions* on page 41
- *Expressions* on page 27
- *NULL Value* on page 75
- *Search Conditions* on page 39
- *Strings* on page 26
- *Three-Valued Logic* on page 53

Disjunction of Subquery Predicates

The SQL89 standard allows for several forms of subquery predicates.

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

The forms of subquery predicates include:

- Unquantified comparison predicates:

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

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

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

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

  The **IN** subquery predicate returns a list of values or a single value. This type is also called a quantified subquery predicate.
- Existence predicates:

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

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

- Quantified comparison predicates:

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

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

The types of queries you can run include:

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

The **SUBQUERY_CACHING_PREFERENCE** option lets experienced DBAs choose which subquery caching method to use.

Examples

Disjunction of uncorrelated **EXISTS** and **IN** subqueries:

```
SELECT COUNT(*)
FROM supplier
WHERE s_suppkey IN (SELECT MAX(l_suppkey)
            FROM lineitem
            GROUP BY l_linenumber)
OR EXISTS (SELECT p_brand
      FROM part
      WHERE p_brand = ‘Brand#43’);
```

Disjunction of uncorrelated **EXISTS** subqueries:

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

Disjunction of uncorrelated scalar or **IN** subquery predicates:

```
SELECT COUNT(*)
FROM supplier
WHERE s_acctbal*10 > (SELECT MAX(o_totalprice)
```

```
             FROM orders
             WHERE o_custkey = 12345)
OR substring(s_name, 1, 6) IN (SELECT c_name
                  FROM Customers
                  WHERE c_nationkey = 10);
```

Disjunction of correlated/uncorrelated quantified comparison subqueries:

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

Disjunction of any correlated subquery predicates:

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

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

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

The following query illustrates a merged query that gets the same results as the example for disjunction of any correlated subquery predicates . Performance of the merged query is suboptimal because it scans the supplier table twice and then merges the results from each **UNION** to return the final result.

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

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

ALL or ANY Conditions

Use ALL or ANY conditions in subqueries in search conditions.

The syntax for **ALL** conditions is:

```
expression compare ALL ( subquery )
```

where *compare* is a comparison operator.

The syntax for **ANY** conditions is:

```
expression compare ANY ( subquery )
```

where *compare* is a comparison operator.

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

```
expression = ANY ( subquery )
```

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

You can use the keyword **SOME** instead of **ANY**.

Restrictions

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

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

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

Compatibility

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

BETWEEN Conditions

Use **BETWEEN** conditions in subqueries to retrieve values within a range.

The syntax for **BETWEEN** conditions is as follows:

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

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

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

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

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

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

LIKE Conditions

Use **LIKE** conditions in subqueries to use wildcards in the WHERE clause to perform pattern matching.

The syntax for **LIKE** conditions is:

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

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

You cannot use subqueries inside a **LIKE** predicate.

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

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

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

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

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

All other characters must match exactly.

For example, the search condition:

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

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

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

Patterns Supported

All patterns of 126 characters or less are supported.

Some patterns between 127 and 254 characters are supported, but only under certain circumstances. See the following subsections for examples.

All patterns 255 characters or greater are not supported.

Patterns Between 127 and 254 Characters: Example 1

Under specific circumstances where adjacent constant characters exist in your pattern, patterns of length between 127 and 254 characters are supported. Each constant character in the string pattern requires two bytes, even if the character is a single-byte character. The string pattern in the **LIKE** predicate must be less than 256 bytes (or 255/2 characters) or else the following error appears:

```
There was an error reading the results of the SQL statement.
The displayed results may be incorrect or incomplete.
Cannot compile Like pattern: either bad pattern or pattern too long.
```

SAP Sybase IQ collapses adjacent constant characters into a single character. For example, consider the following **LIKE** predicate with a string length of 130 characters:

```
select col2 from tablen where col2 like
'1234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456%%%%' ;
```

SAP Sybase IQ collapses the four adjacent constant characters `%%%%` at the end of the string into one `%` character, thereby reducing the length of the string from 130 characters to 127. This is less than the maximum of 256 bytes (or 255/2 characters), and no error is generated.

Therefore, if your LIKE predicate contains adjacent constants in the string, patterns of length between 127 and 254 characters are supported as long as the total length of the collapsed string is less than 256 bytes (or 255/2 characters).

Patterns between 127 and 254 characters: example 2

In this example, the constant characters `7890` replace the four adjacent constant characters `%%%%` at the end of the 130-character **LIKE** predicate:

```
select col2 from tablen where col2 like
'1234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456 7890' ;
```

In this case, no characters are collapsed. The character string length remains at 130 characters and SAP Sybase IQ generates an error.

Patterns Between 127 and 254 Characters: Example 3

In this example, four adjacent underscores ____ (special characters) replace the four constant characters `%%%%` at the end of the 130-character **LIKE** predicate:

```
select col2 from tablen where col2 like
'12345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456____' ;
```

SAP Sybase IQ does not collapse adjacent special characters. The string length remains at 130 characters and SAP Sybase IQ generates an error.

Patterns Between 127 and 254 Characters: Example 4

In this example, the range `[1-3]` replaces the four constant characters `%%%%` at the end of the 130-character **LIKE** predicate:

```
select col2 from tablen where col2 like
'12345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456[1-3]'
;
```

The length of the **LIKE** predicate in bytes is calculated as follows: 126 (for the constant characters) * 2 + 1 (for the `1` in brackets) + 1 (for the `3` in brackets) + 2 (for the Set state and Range state expression).

This equals 256 bytes, and therefore SAP Sybase IQ generates an error.

Searching for One of a Set of Characters

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

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

Searching for One of a Range of Characters

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

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

The range of characters [a-z] is interpreted as “greater than or equal to a, and less than or equal to z,” where the greater than and less than operations are carried out within the collation of the database. For information on ordering of characters within a collation, see *How the Collation Sequence Sorts Characters* in Administration: Globalization.

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

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

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

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

Combining Searches for Ranges and Sets

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

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

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

Searching for One Character Not in a Range

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

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

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

Special Cases of Ranges and Sets

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

Other special cases are:

- The expression *[a-]* matches either of the characters *a* or *-*.
- The expression *[]* is never matched and always returns no rows.
- The expressions *[* or *[abp-q* are ill-formed expressions, and give syntax errors.
- You cannot use wildcard characters inside square brackets. The expression *[a%b]* finds one of *a*, *%*, or *b*.
- You cannot use the caret character to negate ranges except as the first character in the bracket. The expression *[a^b]* finds one of *a*, *^*, or *b*.

Compatibility

The **ESCAPE** clause is supported by SAP Sybase IQ only.

Note: For information on support of the **LIKE** predicate with large object data and variables, see *Unstructured Data Queries* in Unstructured Data Analytics.

Users must be specifically licensed to use the large object data types `LONG BINARY` and `LONG VARCHAR`. For details on the Unstructured Data Analytics Option, see Unstructured Data Analytics.

See also

- *PATINDEX Function [String]* on page 292
- *LOCATE Function [String]* on page 266

IN conditions

Use IN conditions in subqueries to reduce the need to use multiple OR conditions:

The syntax for **IN** conditions is:

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

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

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

Compatibility

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

CONTAINS Conditions

Use CONTAINS conditions in subqueries to define text-matching.

The syntax for **CONTAINS** conditions for a column with a **WD** index is as follows:

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

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

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

For example, this search condition:

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

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

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

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

For information on using **CONTAINS** conditions with **TEXT** indexes, see *Unstructured Data Analytics*.

EXISTS Conditions

An EXISTS condition is met if the subquery returns at least one row.

The syntax for **EXISTS** conditions is as follows:

```
EXISTS( subquery )
```

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

Compatibility

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

IS NULL Conditions

Use IS NULL conditions in subqueries to NULL values represent missing unknown data.

The syntax for **IS NULL** conditions is:

```
expression IS [ NOT ] NULL
```

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

Compatibility

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

Conditions with Logical Operators

Combine search conditions in subqueries using **AND**, **OR**, and **NOT**.

Conditions are combined using **AND** as follows:

```
condition1 AND condition2
```

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

Conditions are combined using **OR** as follows:

```
condition1 OR condition2
```

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

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

NOT Conditions

The NOT condition can be either TRUE, FALSE, or UNKNOWN.

The syntax for **NOT** conditions is:

```
NOT condition1
```

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

Truth Value Conditions

The truth value of a condition is either TRUE or FALSE.

The syntax for truth value conditions is:

```
IS [ NOT ] truth-value
```

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

Compatibility
Truth-valued conditions are supported by SAP Sybase IQ only.

Three-Valued Logic

The AND, OR, NOT, and IS logical operators of SQL work in three-valued logic.

These tables show the three-valued logic.

AND Operator

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

OR Operator

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

NOT Operator

TRUE	FALSE	UNKNOWN
FALSE	TRUE	UNKNOWN

IS Operator

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

See also

- *Comparison Conditions* on page 41
- *Expressions* on page 27
- *NULL Value* on page 75
- *Search Conditions* on page 39
- *Strings* on page 26
- *SQL Operators* on page 30
- *Subqueries in Search Conditions* on page 42

User-Supplied Condition Hints

The selectivity of a condition is the fraction of the table’s rows that satisfy that condition.

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

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

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

User-Supplied Condition Selectivity

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

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

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

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

Examples

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

  ```
  SELECT  ShipDate
  FROM  SalesOrderItems
  WHERE ( ShipDate < '2001/06/30', 1.5 )
  ORDER BY ShipDate DESC
  ```
- The following query estimates that half a percent of the rows satisfy the condition:

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

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

Compatibility

SQL Anywhere supports user-supplied selectivity estimates.

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

See also

- *User-Supplied Condition Hint Strings* on page 56
- *User-Supplied Hints on Join Equality Conditions* on page 61

- *Guidelines for Usage of User-Supplied Condition Hints* on page 63
- *Selectivity Hints* on page 56

User-Supplied Condition Hint Strings

You can supply additional hint information to the optimizer through a condition hint string.

These per-condition hint strings let users specify additional execution preferences for a condition, which the optimizer follows, if possible. These preferences include which index to use for the condition, the selectivity of the condition, the phase of execution when the condition is executed, and the usefulness of the condition, which affects its ordering among the set of conditions executed within one phase of execution.

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

See also

- *User-Supplied Condition Selectivity* on page 55
- *User-Supplied Hints on Join Equality Conditions* on page 61
- *Guidelines for Usage of User-Supplied Condition Hints* on page 63

Selectivity Hints

The first hint type that can appear within a hint string is a selectivity hint. A selectivity hint is identified by a hint type identifier of either “S” or “s”.

Like user-supplied selectivity estimates, the selectivity value is always expressed as a percentage of the table’s rows, which satisfy the condition.

Example

The following example is exactly equivalent to the second user-supplied condition selectivity example.

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

See also

- *Index Preference Hints* on page 57
- *INDEX_PREFERENCE Option* on page 57
- *Execution Phase Hints* on page 59
- *Usefulness Hints* on page 60

- *User-Supplied Condition Selectivity* on page 55

Index Preference Hints

The second supported hint type is an index preference hint, which is identified by a hint type identifier of either “I” or “i”.

The value for an index preference hint can be any integer between -10 and 10. The meaning of each positive integer value is to prefer a specific index type, while negative values indicate that the specific index type is to be avoided.

The effect of an index preference hint is the same as that of the **INDEX_PREFERENCE** option, except that the preference applies only to the condition it is associated with rather than all conditions within the query. An index preference can only affect the execution of a condition if the specified index type exists on that column and that index type is valid for use when evaluating the associated condition; not all index types are valid for use with all conditions.

Example

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

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

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

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

See also

- *Selectivity Hints* on page 56
- *INDEX_PREFERENCE Option* on page 57
- *Execution Phase Hints* on page 59
- *Usefulness Hints* on page 60

INDEX_PREFERENCE Option

Controls the choice of indexes to use for queries.

Allowed Values

Value	Action
0	Let the optimizer choose

Value	Action
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

Default
0

Scope
Option can be set at the database (PUBLIC) or user level. When set at the database level, the value becomes the default for any new user, but has no impact on existing users. When set at the user level, overrides the PUBLIC value for that user only. No system privilege is required to set option for self. System privilege is required to set at database level or at user level for any user other than self.

Requires the SET ANY PUBLIC OPTION system privilege to set this option. Can be set temporary for an individual connection or for the PUBLIC role. Takes effect immediately.

Description
The SAP 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.

See also

- *Selectivity Hints* on page 56
- *Index Preference Hints* on page 57
- *Execution Phase Hints* on page 59
- *Usefulness Hints* on page 60

Execution Phase Hints

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

Within the SAP Sybase IQ query engine, there are distinct phases of execution where conditions can be evaluated:

- invariant
- delayed
- bound
- horizontal

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

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

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

- **Invariant –** A condition that refers to only one column (or two columns from the same table) and that can be evaluated using an index is generally referred to as a simple invariant condition. Simple invariant condition are normally evaluated early within the optimization process. This means that the number of rows satisfying all of those invariant conditions is available to guide the optimizer’s decisions on the best join order and join algorithms to use. Because this is the earliest phase of execution, a user can never force a condition into this phase, but conditions can be forced out of this phase into later phases.
- **Delayed –** Some conditions cannot be evaluated until some other part of a query has been executed. These delayed conditions are evaluated once when the query node to which they are attached is first fetched. These conditions fall into two categories, uncorrelated

subquery conditions and IN or PROBABLY_IN pushdown join conditions created by the optimizer.

- **Bound –** Some conditions must be evaluated multiple times. These conditions generally fall into two categories: conditions containing outer references within a correlated subquery, and pushdown equality join conditions created by the optimizer. The outer reference conditions, for example, are reevaluated each time the outer reference value changes during the query's execution.
- **Horizontal –** Some conditions, such as those which contain more than two columns from a table, must be evaluated one row at a time, rather than by using an index.

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

- D – Delayed
- B – Bound
- H – Horizontal

Example

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

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

See also

- *Selectivity Hints* on page 56
- *Index Preference Hints* on page 57
- *INDEX_PREFERENCE Option* on page 57
- *Usefulness Hints* on page 60

Usefulness Hints

The final supported hint type is the usefulness hint, which is identified by a hint type identifier of either “U” or “u”.

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

Example

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

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

Compatibility

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

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

See also

- *Selectivity Hints* on page 56
- *Index Preference Hints* on page 57
- *INDEX_PREFERENCE Option* on page 57
- *Execution Phase Hints* on page 59

User-Supplied Hints on Join Equality Conditions

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

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

Example

The following example requests a hash join:

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

Table 5. 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

Value	Action
5	Prefer hash push-down
6	Prefer asymmetric sort-merge join
7	Prefer sort-merge push-down
8	Prefer asymmetric sort-merge push-down join
9	Prefer partitioned hash join if the join keys include all the partition keys of a hash partitioned table
10	Prefer partitioned hash-push down join if the join keys include all the partition keys of a hash partitioned table
11	Prefer partitioned sort-merge join if the join keys include all the partition keys of a hash partitioned table
12	Prefer partitioned sort-merge push-down join if the join keys include all the partition keys of a hash partitioned table
-1	Avoid sort-merge
-2	Avoid nested-loop
-3	Avoid nested-loop push-down
-4	Avoid hash
-5	Avoid hash push-down
-6	Avoid asymmetric sort-merge join
-7	Avoid sort-merge push-down
-8	Avoid asymmetric sort-merge push-down join
-9	Avoid partitioned hash join if the join keys include all the partition keys of a hash partitioned table
-10	Avoid partitioned hash-push down join if the join keys include all the partition keys of a hash partitioned table
-11	Avoid partitioned sort-merge join if the join keys include all the partition keys of a hash partitioned table
-12	Avoid partitioned sort-merge push-down join if the join keys include all the partition keys of a hash partitioned table

For more information
Reference: Statements and Options > Database Options > Alphabetical List of Options > JOIN_PREFERENCE Option

See also
- *User-Supplied Condition Selectivity* on page 55
- *User-Supplied Condition Hint Strings* on page 56
- *Guidelines for Usage of User-Supplied Condition Hints* on page 63

Guidelines for Usage of User-Supplied Condition Hints

Condition hints are generally appropriate only within frequently run queries.

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

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

See also
- *User-Supplied Condition Selectivity* on page 55
- *User-Supplied Condition Hint Strings* on page 56
- *User-Supplied Hints on Join Equality Conditions* on page 61

Special Values

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

See also
- *Expressions* on page 27

CURRENT DATABASE Special Value

CURRENT DATABASE returns the name of the current database.

Data Type
`STRING`

CURRENT DATE Special Value

CURRENT DATE returns the current year, month and day.

Data Type

`DATE`

See also

- *TIMESTAMP Special Value* on page 66
- *CURRENT TIMESTAMP Special Value* on page 65
- *CURRENT TIME Special Value* on page 64
- *Date and Time Data Types* on page 400
- *Retrieve Dates and Times* on page 403

CURRENT PUBLISHER Special Value

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

Data Type

`STRING`

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

CURRENT TIME Special Value

CURRENT TIME returns the current hour, minute, second, and fraction of a second.

Data Type

`TIME`

Description

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

See also

- *TIMESTAMP Special Value* on page 66
- *CURRENT TIMESTAMP Special Value* on page 65
- *CURRENT DATE Special Value* on page 64
- *Date and Time Data Types* on page 400
- *Retrieve Dates and Times* on page 403

CURRENT TIMESTAMP Special Value

Combines **CURRENT DATE** and **CURRENT TIME** to form a **TIMESTAMP** value containing the year, month, day, hour, minute, second and fraction of a second.

As with **CURRENT TIME**, the accuracy of the fraction of a second is limited by the system clock.

CURRENT TIMESTAMP defaults to 3 digits.

Data type
`TIMESTAMP`

See also

- *TIMESTAMP Special Value* on page 66
- *CURRENT TIME Special Value* on page 64
- *CURRENT DATE Special Value* on page 64
- *Date and Time Data Types* on page 400
- *Retrieve Dates and Times* on page 403
- *CURRENT USER Special Value* on page 65
- *LAST USER Special Value* on page 65
- *USER Special Value* on page 67

CURRENT USER Special Value

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

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

Data Type
`STRING`

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

See also

- *CURRENT TIMESTAMP Special Value* on page 65
- *LAST USER Special Value* on page 65
- *USER Special Value* on page 67

LAST USER Special Value

LAST USER returns the name of the user who last modified the row.

On **INSERT** and **LOAD**, this constant has the same effect as **CURRENT USER**. On **UPDATE**, if a column with a default value of **LAST USER** is not explicitly modified, it is changed to the name of the current user.

When combined with the **DEFAULT TIMESTAMP**, a default value of **LAST USER** can be used to record (in separate columns) both the user and the date and time a row was last changed.

Data Type
`STRING`

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

See also

- *CURRENT USER Special Value* on page 65
- *CURRENT TIMESTAMP Special Value* on page 65
- *USER Special Value* on page 67

SQLCODE Special Value

SQLCODE returns the current **SQLCODE** value.

The **SQLCODE** value is set after each statement. You can check the **SQLCODE** to see whether or not the statement succeeded.

DATA TYPE
`STRING`

SQLSTATE Special Value

SQLSTATE returns the current **SQLSTATE** value.

The **SQLSTATE** value is set after each statement. You can check the **SQLSTATE** to see whether or not the statement succeeded.

Data Type
`STRING`

TIMESTAMP Special Value

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

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

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

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

Data Type
`TIMESTAMP`

See also
- *CURRENT TIMESTAMP Special Value* on page 65
- *CURRENT TIME Special Value* on page 64
- *CURRENT DATE Special Value* on page 64
- *Date and Time Data Types* on page 400
- *Retrieve Dates and Times* on page 403

USER Special Value

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

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

Data Type
`STRING`

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

See also
- *CURRENT USER Special Value* on page 65
- *CURRENT TIMESTAMP Special Value* on page 65
- *LAST USER Special Value* on page 65

Variables

SAP Sybase IQ supports local variables, connection-level variables, and global variables.

All global variables have names beginning with two @ signs. For example, the global variable *@@version* has a value that is the current version number of the database server. Users cannot define global variables.

Local Variables

Local variables are declared by the user, and can be used in procedures or in batches of SQL statements to hold information.

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

The syntax of the **DECLARE** statement is as follows:

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

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

Examples

- The following batch illustrates the use of local variables:

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

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

  ```
  local_var = 10
  ```
- The variable `local_var` does not exist outside the compound statement in which it is declared. The following batch is invalid, and displays a column not found error:

  ```
  -- This batch is invalid.
  BEGIN
      DECLARE local_var INT ;
      SET local_var = 10 ;
      MESSAGE 'local_var = ', local_var ;
  END;
  MESSAGE 'local_var = ', local_var ;
  ```
- The following example illustrates the use of **SELECT** with an **INTO** clause to set the value of a local variable:

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

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

  ```
  local_var = 10
  ```

Compatibility

Names—Adaptive Server Enterprise and SAP Sybase IQ both support local variables. In Adaptive Server Enterprise, all variables must be prefixed with an @ sign. In SAP Sybase IQ, the @ prefix is optional. To write compatible SQL, ensure all your variables have the @ prefix.

Scope—The scope of local variables differs between SAP Sybase IQ and Adaptive Server Enterprise. SAP Sybase IQ supports the use of the **DECLARE** statement to declare local variables within a batch. However, if the **DECLARE** is executed within a compound statement, the scope is limited to the compound statement.

Declaration—Only one variable can be declared for each **DECLARE** statement in SAP Sybase IQ. In Adaptive Server Enterprise, more than one variable can be declared in a single statement.

Connection-Level Variables

Connection-level variables are declared by the user, and can be used in procedures or in batches of SQL statements to hold information.

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

The syntax for **CREATE VARIABLE** is:

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

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

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

```
DROP VARIABLE con_var
```

Example

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

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

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

  ```
  con_var = 10
  ```

Compatibility

Adaptive Server Enterprise does not support connection-level variables.

Global Variables

Global variables are system-supplied variables that provide system-supplied values.

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

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

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

Global Variable and Special Constants

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

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

```
SELECT @@version
```

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

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

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

List of Global Variables

This table lists the global variables available in SAP Sybase IQ.

Variable name	Meaning
@@error	Commonly used to check the error status (succeeded or failed) of the most recently executed statement. Contains 0 if the previous transaction succeeded; otherwise, contains the last error number generated by the system. A statement such as if @@error != 0 return causes an exit if an error occurs. Every SQL statement resets @@error, so the status check must immediately follow the statement whose success is in question.
@@fetch_status	Contains status information resulting from the last fetch statement. @@fetch_status may contain the following values • 0 The fetch statement completed successfully. • -1 The fetch statement resulted in an error. • -2 There is no more data in the result set. This feature is the same as @@sqlstatus, except that it returns different values. It is for Microsoft SQL Server compatibility.

Variable name	Meaning
@@identity	The last value inserted into an Identity/Autoincrement column by an insert, load or update statement. *@@identity* is reset each time a row is inserted into a table. If a statement inserts multiple rows, *@@identity* reflects the Identity/Autoincrement value for the last row inserted. If the affected table does not contain an Identity/Autoincrement column, *@@identity* is set to 0. The value of *@@identity* is not affected by the failure of an insert, load, or update statement, or the rollback of the transaction that contained the failed statement. *@@identity* retains the last value inserted into an Identity/Autoincrement column, even if the statement that inserted that value fails to commit.
@@isolation	Current isolation level. @@isolation takes the value of the active level.
@@procid	Stored procedure ID of the currently executing procedure.
@@rowcount	Number of rows affected by the last statement. The value of *@@rowcount* should be checked immediately after the statement. Inserts, updates, and deletes set *@@rowcount* to the number of rows affected. With cursors, *@@rowcount* represents the cumulative number of rows returned from the cursor result set to the client, up to the last fetch request. The *@@rowcount* is not reset to zero by any statement which does not affect rows, such as an **IF** statement.
@@servername	Name of the current database server.
@@sqlstatus	Contains status information resulting from the last **FETCH** statement.
@@version	Version number of the current version of SAP Sybase IQ.

Adaptive Server Enterprise Global Variables Supported in SAP Sybase IQ

This table includes all Adaptive Server Enterprise global variables that are supported in SAP Sybase IQ. Adaptive Server Enterprise global variables that are not supported by SAP Sybase IQ are not included in the list.

This list includes all global variables that return a value, including those for which the value is fixed at NULL, 1, -1, or 0, and might not be meaningful.

Table 6. Adaptive Server Enterprise global variables supported in SAP Sybase IQ

Global variable	Returns
@@char_convert	Returns 0.

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

Global variable	Returns
@@langid	In Adaptive Server Enterprise, defines the local language ID of the language currently in use. In SAP Sybase IQ, returns 0.
@@language	In Adaptive Server Enterprise, defines the name of the language currently in use. In SAP Sybase IQ, returns "English".
@@maxcharlen	In Adaptive Server Enterprise, maximum length, in bytes, of a character in the Adaptive Server Enterprise default character set. In SAP Sybase IQ, returns 1.
@@max_ connections	For the network server, the maximum number of active clients (not database connections, as each client can support multiple connections). For Adaptive Server Enterprise, the maximum number of connections to the server.
@@ncharsize	In Adaptive Server Enterprise, average length, in bytes, of a national character. In SAP Sybase IQ, returns 1.
@@nestlevel	In Adaptive Server Enterprise, nesting level of current execution (initially 0). Each time a stored procedure or trigger calls another stored procedure or trigger, the nesting level is incremented. In SAP Sybase IQ, returns -1.
@@pack_received	In Adaptive Server Enterprise, number of input packets read by Adaptive Server Enterprise since the server was last started. In SAP Sybase IQ, returns 0.
@@pack_sent	In Adaptive Server Enterprise, number of output packets written by Adaptive Server Enterprise since the server was last started. In SAP Sybase IQ, returns 0.
@@packet_errors	In Adaptive Server Enterprise, number of errors that have occurred while Adaptive Server Enterprise was sending and receiving packets. In SAP Sybase IQ, returns 0.
@@procid	Stored procedure ID of the currently executing procedure.
@@servername	Name of the local Adaptive Server Enterprise or SAP Sybase IQ server.
@@spid	In Adaptive Server Enterprise, server process ID number of the current process. In SAP Sybase IQ, the connection handle for the current connection. This is the same value as that displayed by the **sa_conn_info** procedure.
@@sqlstatus	Contains status information resulting from the last **FETCH** statement. *@@sqlstatus* may contain the following values: • 0 – the FETCH statement completed successfully. • 1 – the FETCH statement resulted in an error. • 2 – there is no more data in the result set.

Global variable	Returns
@@thresh_hysteresis	In Adaptive Server Enterprise, change in free space required to activate a threshold. In SAP Sybase IQ, returns 0.
@@timeticks	In Adaptive Server Enterprise, number of microseconds per tick. The amount of time per tick is machine-dependent. In SAP Sybase IQ, returns 0.
@@total_errors	In Adaptive Server Enterprise, number of errors that have occurred while Adaptive Server Enterprise was reading or writing. In SAP Sybase IQ, returns 0.
@@total_read	In Adaptive Server Enterprise, number of disk reads by Adaptive Server Enterprise since the server was last started. In SAP Sybase IQ, returns 0.
@@total_write	In Adaptive Server Enterprise, number of disk writes by Adaptive Server Enterprise since the server was last started. In SAP Sybase IQ, returns 0.
@@tranchained	Current transaction mode of the Transact-SQL program. *@@tranchained* returns 0 for unchained or 1 for chained.
@@trancount	Nesting level of transactions. Each **BEGIN TRANSACTION** in a batch increments the transaction count.
@@transtate	In Adaptive Server Enterprise, current state of a transaction after a statement executes. In SAP Sybase IQ, returns -1.
@@version	Information about the current version of Adaptive Server Enterprise or SAP Sybase IQ.

Comments

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

These comment indicators are available in SAP Sybase IQ:

Comment Indicator	Description
-- (Double hyphen)	The database server ignores any remaining characters on the line. This is the SQL92 comment indicator.
// (Double slash)	The double slash has the same meaning as the double hyphen.

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

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

Examples

This example illustrates the use of double-dash comments:

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

This example illustrates the use of C-style comments:

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

NULL Value

Use NULL to specify a value that is unknown, missing, or not applicable.

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

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

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

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

You can also use the **IS [NOT]** *truth-value* clause, where *truth-value* is one of TRUE, FALSE or UNKNOWN, to select rows where the NULL value is involved.

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

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

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

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

Many common errors in formulating SQL queries are caused by the behavior of NULL. Be careful to avoid these problem areas. Note the effect of three-valued logic when combining search conditions.

Syntax

```
NULL
```

Usage
Anywhere

Permissions
Must be connected to the database

Side Effects
None

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

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

See also

- *Comparison Conditions* on page 41
- *Expressions* on page 27
- *Search Conditions* on page 39
- *Strings* on page 26
- *Three-Valued Logic* on page 53
- *SQL Operators* on page 30
- *Subqueries in Search Conditions* on page 42

SQL Functions

Functions return information from the database and are allowed anywhere an expression is allowed.

When using functions with SAP Sybase IQ, unless otherwise stated, any function that receives the NULL value as a parameter returns a NULL value.

If you omit the **FROM** clause, or if all tables in the query are in the SYSTEM dbspace, SQL Anywhere processes the query, instead of SAP Sybase IQ, and might behave differently, especially with regard to syntactic and semantic restrictions and the effects of option settings.

If you have a query that does not require a **FROM** clause, you can force SAP Sybase IQ to process the query by adding the clause "FROM iq_dummy," where `iq_dummy` is a one-row, one-column table that you create in your database.

See also

- *Additional Functions* on page 417

Aggregate Functions

Aggregate functions summarize data over a group of rows from the database. The groups are formed using the **GROUP BY** clause of the **SELECT** statement.

Usage
Simple aggregate functions, such as **SUM()**, **MIN()**, **MAX()**, **AVG()** and **COUNT()** are allowed only in the select list and in the **HAVING** and **ORDER BY** clauses of a **SELECT** statement. These functions summarize data over a group of rows from the database. Groups are formed using the **GROUP BY** clause of the **SELECT** statement.

A new class of aggregate functions, called *window functions*, provides moving averages and cumulative measures that compute answers to queries such as, "What is the quarterly moving average of the Dow Jones Industrial average," or "List all employees and their cumulative salaries for each department."

- Simple aggregate functions, such as **AVG()**, **COUNT()**, **MAX()**, **MIN()**, and **SUM()** summarize data over a group of rows from the database. The groups are formed using the **GROUP BY** clause of the **SELECT** statement.
- Newer statistical aggregate functions that take one argument include **STDDEV()**, **STDDEV_SAMP()**, **STDDEV_POP()**, **VARIANCE()**, **VAR_SAMP()**, and **VAR_POP().**

Both the simple and newer categories of aggregates can be used as a windowing function that incorporates a **<window clause>** in a SQL query specification (a *window*) that conceptually creates a moving window over a result set as it is processed.

Another class of window aggregate functions supports analysis of time series data. Like the simple aggregate and statistical aggregate functions, you can use these window aggregates with a SQL query specification (or *window-spec*). The time series window aggregate functions calculate correlation, linear regression, ranking, and weighted average results:

- ISO/ANSI SQL:2008 OLAP functions for time series analysis include: **CORR**(), **COVAR_POP()**, **COVAR_SAMP()**, **CUME_DIST()**, **FIRST_VALUE()**, **LAST_VALUE**(), **REGR_AVGX()**, **REGR_AVGY()**, **REGR_COUNT()**, **REGR_INTERCEPT()**, **REGR_R2()**, **REGR_SLOPE()**, **REGR_SXX()**, **REGR_SXY**(), and **REGR_SYY()**.
- Non-ISO/ANSI SQL:2008 OLAP aggregate function extensions used in the database industry include **FIRST_VALUE**(), **MEDIAN()**, and **LAST_VALUE()**.
- Weighted OLAP aggregate functions that calculate weighted moving averages include **EXP_WEIGHTED_AVG()** and **WEIGHTED_AVG()**.

Time series functions designed exclusively for financial time series forecasting and analysis have names beginning with "**TS_**".

For more information about OLAP, see *Appendix: Using OLAP* in Programming.

For information on aggregate function support of the `LONG BINARY` and `LONG VARCHAR` data types, see Unstructured Data Analytics.

Table 7. Aggregate functions

Aggregate function	Parameters
AVG	([DISTINCT] { *column-name* | *numeric-expr* })
CORR	(dependent-expression, independent-expression)
COUNT	(*)
COUNT	([DISTINCT] { *column-name* | *numeric-expr* })
COVAR_POP	(dependent-expression, independent-expression)
COVAR_SAMP	(dependent-expression, independent-expression)
CUME_DIST	( )
EXP_WEIGHTED_AVG	(*expression*, *period-expression*)
FIRST_VALUE	(expression)
LAST_VALUE	(expression)
LIST	([DISTINCT] *string-expression* [, *'delimiter-string'*] [ORDER BY *order-by-expression* [ASC | DESC], ...)
MAX	([DISTINCT] { *column-name* | *numeric-expr* })

Aggregate function	Parameters
MEDIAN	(expression)
MIN	([DISTINCT] { *column-name* | *numeric-expr* })
REGR_AVGX	(dependent-expression, independent-expression)
REGR_AVGY	(dependent-expression, independent-expression)
REGR_COUNT	(dependent-expression, independent-expression)
REGR_INTERCEPT	(dependent-expression, independent-expression)
REGR_R2	(dependent-expression, independent-expression)
REGR_SLOPE	(dependent-expression, independent-expression)
REGR_SXX	(dependent-expression, independent-expression)
REGR_SXY	(dependent-expression, independent-expression)
REGR_SYY	(dependent-expression, independent-expression)
STDDEV	([ALL] *expression*)
SUM	([DISTINCT] { *column-name* | *numeric-expr* })
VARIANCE	([ALL] *expression*)
WEIGHTED_AVG	(*expression*, *period-expression*

The aggregate functions **AVG**, **SUM**, **STDDEV**, and **VARIANCE** do not support the binary data types (`BINARY` and `VARBINARY`).

See also

- *Analytical Functions* on page 81

Analytical Functions

Analytical functions include simple aggregates, window functions, and numeric functions.

- Simple aggregates — **AVG**, **COUNT**, **MAX**, **MIN**, and **SUM**, **STDDEV**, and **VARIANCE**

  Note: You can use all simple aggregates, except the Grouping() function, with an OLAP windowed function.

- Window functions:
  - Windowing aggregates — **AVG**, **COUNT**, **MAX**, **MIN**, and **SUM**.

    - Ranking functions — **RANK**, **DENSE_RANK**, **PERCENT_RANK**, **ROW_NUMBER**, and **NTILE**.
    - Statistical functions — **STDDEV**, **STDDEV_SAMP**, **STDDEV_POP**, **VARIANCE**, **VAR_SAMP**, and **VAR_POP**.
    - Distribution functions — **PERCENTILE_CONT** and **PERCENTILE_DISC**.
    - Inter-row functions — **LAG** and **LEAD**.
- Numeric functions — **WIDTH_BUCKET**, **CEIL**, and **LN**, **EXP**, **POWER**, **SQRT**, and **FLOOR**.

Note: The ranking and inverse distribution analytical functions are not supported by Adaptive Server Enterprise.

This table lists the analytical functions and their parameters. Unlike some aggregate functions, you cannot specify **DISTINCT** in window functions.

Table 8. Analytical functions

Function	Parameters
AVG	({ *column-name* | *numeric-expr* })
COUNT	(*)
COUNT	({ *column-name* | *expression* })
DENSE_RANK	()
GROUPING *	({ GROUPING *group-by-expression* **})**
MAX	({ *column-name* | *expression* })
MIN	({ *column-name* | *expression* })
NTILE	(*integer*)
PERCENT_RANK	()
PERCENTILE_CONT	(*numeric-expr*)
PERCENTILE_DISC	(*numeric-expr*)
RANK	()
ROW_NUMBER	()
STDDEV	([ALL] *expression*)
STDDEV_POP	([ALL] *expression*)
STDDEV_SAMP	([ALL] *expression*)
SUM	({ *column-name* | *expression* })
VAR_POP	([ALL] *expression*)

Function	Parameters
VAR_SAMP	([ALL] *expression*)
VARIANCE	([ALL] *expression*)

* The OLAP SQL standard allows Grouping() in **GROUP BY CUBE**, or **GROUP BY ROLLUP** operations only.

See also

- *Aggregate Functions* on page 79

Windowing Aggregate Function Usage

A major feature of the ISO/ANSI SQL extensions for OLAP is a construct called a *window*. This windowing extension let users divide result sets of a query (or a logical partition of a query) into groups of rows called partitions and determine subsets of rows to aggregate with respect to the current row.

You can use three classes of window functions with a window: ranking functions, the row numbering function, and window aggregate functions.

Windowing extensions specify a window function type over a window name or specification and are applied to partitioned result sets within the scope of a single query expression.

Windowing operations let you establish information such as the ranking of each row within its partition, the distribution of values in rows within a partition, and similar operations. Windowing also lets you compute moving averages and sums on your data, enhancing the ability to evaluate your data and its impact on your operations.

A window partition is a subset of rows returned by a query, as defined by one or more columns in a special **OVER()** clause:

```
OVER (PARTITION BY col1, col2...)
```

For information on analytical function support of the LONG BINARY and LONG VARCHAR data types, see *Funtion Support* in Unstructured Data Analytics.

See also

- *CORR Function [Aggregate]* on page 182
- *COUNT Function [Aggregate]* on page 187
- *EXP_WEIGHTED_AVG Function [Aggregate]* on page 222
- *FIRST_VALUE Function [Aggregate]* on page 224
- *GROUPING Function [Aggregate]* on page 230
- *LAST_VALUE Function [Aggregate]* on page 254
- *MAX Function [Aggregate]* on page 271
- *MEDIAN Function [Aggregate]* on page 272

- *MIN Function [Aggregate]* on page 273
- *REGR_AVGX Function [Aggregate]* on page 308
- *REGR_COUNT Function [Aggregate]* on page 311
- *REGR_INTERCEPT Function [Aggregate]* on page 312
- *REGR_R2 Function [Aggregate]* on page 313
- *REGR_SLOPE Function [Aggregate]* on page 314
- *REGR_SXX Function [Aggregate]* on page 315
- *REGR_SXY Function [Aggregate]* on page 317
- *REGR_SYY Function [Aggregate]* on page 318
- *STDDEV Function [Aggregate]* on page 347
- *SUM Function [Aggregate]* on page 360
- *STDDEV_POP Function [Aggregate]* on page 349
- *STDDEV_SAMP Function [Aggregate]* on page 350
- *VAR_POP Function [Aggregate]* on page 370
- *VAR_SAMP Function [Aggregate]* on page 372
- *VARIANCE Function [Aggregate]* on page 373
- *WEIGHTED_AVG Function [Aggregate]* on page 376

Ranking Functions Usage

The OLAP ranking functions let application developers compose single-statement SQL queries that answer questions such as “Name the top 10 products shipped this year by total sales,” or “Give the top 5% of salespeople who sold orders to at least 15 different companies.”

These functions include the ranking functions, **RANK()**, **DENSE_RANK()**, **PERCENT_RANK()**, **ROW_NUMBER()**, and **NTILE()**.

Rank analytical functions rank items in a group, compute distribution, and divide a result set into a number of groupings. The rank analytical functions, **RANK()**, **DENSE_RANK()**, **PERCENT_RANK()**, **ROW_NUMBER()**, and **NTILE()** all require an **OVER** (**ORDER BY**) clause. For example:

```
RANK() OVER ( [PARTITION BY] ORDER BY <expression>
[ ASC | DESC ] )
```

The **ORDER BY** clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the OVER clause and is not an **ORDER BY** for **SELECT**. No aggregation functions in the rank query **ROW** are allowed to specify **DISTINCT**.

Note: The **OVER** (**ORDER_BY**) clause of the **ROW_NUMBER()** function cannot contain a **ROWS** or **RANGE** clause.

The **OVER** clause indicates that the function operates on a query result set. The result set is the rows that are returned after the **FROM**, **WHERE**, **GROUP BY**, and **HAVING** clauses have all been evaluated. The **OVER** clause defines the data set of the rows to include in the computation of the rank analytical function.

The value *expression* is a sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

The ASC or DESC parameter specifies the ordering sequence as ascending or descending. Ascending order is the default.

Rank analytical functions are only allowed in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. Rank functions can be in a view or a union. You cannot use rank functions in a subquery, a **HAVING** clause, or in the select list of an **UPDATE** or **DELETE** statement. More than one rank analytical function is allowed per query in SAP Sybase IQ 16.0.

Statistical Aggregate Analytic Function Usage

Statistical aggregate analytic functions summarize data over a group of rows from the database.

The groups are formed using the **GROUP BY** clause of the **SELECT** statement. Aggregate functions are allowed only in the select list and in the **HAVING** and **ORDER BY** clauses of a **SELECT** statement. These functions include **STDDEV**, **STDDEV_POP**, **STDDEV_SAMP**, **VARIANCE**, **VAR_POP**, and **VAR_SAMP**.

The OLAP functions can be used as a window function with an **OVER()** clause in a SQL query specification that conceptually creates a moving window over a result set as it is processed.

Distribution Functions Usage

The inverse distribution analytical functions **PERCENTILE_CONT** and **PERCENTILE_DISC** take a percentile value as the function argument and operate on a group of data specified in the **WITHIN GROUP** clause, or operate on the entire data set.

These functions return one value per group. For **PERCENTILE_DISC**, the data type of the results is the same as the data type of its **ORDER BY** item specified in the **WITHIN GROUP** clause. For **PERCENTILE_CONT**, the data type of the results is either numeric, if the **ORDER BY** item in the **WITHIN GROUP** clause is a numeric, or double, if the **ORDER BY** item is an integer or floating-point.

The inverse distribution analytical functions require a **WITHIN GROUP** (**ORDER BY**) clause. For example:

```
PERCENTILE_CONT ( expression1 ) WITHIN GROUP ( ORDER BY expression2
[ASC | DESC ] )
```

The value of *expression1* must be a constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, then a “wrong argument for percentile” error is returned. If the argument value is less than 0, or greater than 1, then a “data value out of range” error is returned.

The **ORDER BY** clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the **WITHIN GROUP** clause and is not an **ORDER BY** for the **SELECT**.

The **WITHIN GROUP** clause distributes the query result into an ordered data set from which the function calculates a result.

The value *expression2* is a sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

The ASC or DESC parameter specifies the ordering sequence as ascending or descending. Ascending order is the default.

Inverse distribution analytical functions are allowed in a subquery, a **HAVING** clause, a view, or a union. The inverse distribution functions can be used anywhere the simple non analytical aggregate functions are used. The inverse distribution functions ignore the NULL value in the data set.

Inter-Row Functions Usage

The inter-row functions **LAG** and **LEAD** enable access to previous values or subsequent values in a data series.

These functions provide access to more than one row of a table or partition simultaneously without a self join. The **LAG** function provides access to a row at a given physical offset prior to the **CURRENT ROW** in the table or partition. The **LEAD** function provides access to a row at a given physical offset after the **CURRENT ROW** in the table or partition. Use the **LAG** and **LEAD** functions to create queries such as “What was the stock price two intervals before the current row,” and “What was the stock price one interval after the current row.”

Inter-row functions require an **OVER** (**ORDER_BY**) clause.

Inter-Row Functions

The inter-row functions, **LAG** and **LEAD**, provide access to previous or subsequent values in a data series, or to multiple rows in a table.

Inter-row functions also partition simultaneously without a self-join. LAG provides access to a row at a given physical offset prior to the **CURRENT ROW** in the table or partition. LEAD provides access to a row at a given physical offset after the **CURRENT ROW** in the table or partition.

LAG and **LEAD** syntax is identical. Both functions require an **OVER** (**ORDER_BY**) window specification. For example:

```
LAG (value_expr) [, offset [, default]]) OVER ([PARTITION BY window
partition] ORDER BY window ordering)
```

and:

LEAD (*value_expr*) [, *offset* [, *default*]]) **OVER** ([**PARTITION BY** *window partition*] **ORDER BY** *window ordering*)

The **PARTITION BY** clause in the **OVER** (**ORDER_BY**) clause is optional. The **OVER** (**ORDER_BY**) clause cannot contain a window frame **ROWS**/**RANGE** specification.

value_expr is a table column or expression that defines the offset data to return from the table. You can define other functions in the *value_expr*, with the exception of analytic functions.

For both functions, specify the target row by entering a physical offset. The *offset* value is the number of rows above or below the current row. Enter a nonnegative numeric data type (entering a negative value generates an error). If you enter 0, SAP Sybase IQ returns the current row.

The optional *default* value defines the value to return if the *offset* value goes beyond the scope of the table. The default value of *default* is **NULL**. The data type of *default* must be implicitly convertible to the data type of the *value_expr* value, or SAP Sybase IQ generates a conversion error.

LAG example 1—The inter-row functions are useful in financial services applications that perform calculations on data streams, such as stock transactions. This example uses the **LAG** function to calculate the percentage change in the trading price of a particular stock. Consider the following trading data from a fictional table called `stock_trades`:

```
traded at            symbol   price
-------------------  ------   ------
2009-07-13 06:07:12  SQL      15.84
2009-07-13 06:07:13  TST       5.75
2009-07-13 06:07:14  TST       5.80
2009-07-13 06:07:15  SQL      15.86
2009-07-13 06:07:16  TST       5.90
2009-07-13 06:07:17  SQL      15.86
```

Note: The fictional `stock_trades` table is not available in the `iqdemo` database.

The query partitions the trades by stock symbol, orders them by time of trade, and uses the **LAG** function to calculate the percentage increase or decrease in trade price between the current trade and the previous trade:

```
select  stock_symbol as 'Stock',
    traded_at    as 'Date/Time of Trade',
    trade_price  as 'Price/Share',
    cast ( ( ( (trade_price
        - (lag(trade_price, 1)
        over (partition by stock_symbol
            order by traded_at)))
        / trade_price)
    * 100.0) as numeric(5, 2) )
        as '% Price Change vs Previous Price'
from stock_trades
order by 1, 2
```

The query returns these results:

```
Stock   Date/Time of Trade   Price/  % Price Change_vs
symbol                       Share   Previous Price
------  -------------------  -----   -----------------
SQL     2009-07-13 06:07:12  15.84   NULL
SQL     2009-07-13 06:07:15  15.86   0.13
SQL     2009-07-13 06:07:17  15.86   0.00
TST     2009-07-13 06:07:13   5.75   NULL
TST     2009-07-13 06:07:14   5.80   0.87
TST     2009-07-13 06:07:16   5.90   1.72
```

The `NULL` result in the first and fourth output rows indicates that the **LAG** function is out of scope for the first row in each of the two partitions. Since there is no previous row to compare to, SAP Sybase IQ returns `NULL` as specified by the *default* variable.

Data Type Conversion Functions

Data type conversion functions convert arguments from one data type to another.

This table lists the data type conversion functions and their parameters.

Table 9. Data type conversion functions

Data type conversion function	Parameters
BIGINTTOHEX	(*integer-expression*)
CAST	(*expression* **AS** *data type*)
CONVERT	(*data type*, *expression* [, *format-style*])
HEXTOBIGINT	(*hexadecimal-string*)
HEXTOINT	(*hexadecimal-string*)
INTTOHEX	(*integer-expr*)
ISDATE	(*string*)
ISNUMERIC	(*string*)

Description

The database server carries out many type conversions automatically. For example, if a string is supplied where a numerical expression is required, the string is automatically converted to a number.

See also

- *Data Type Conversions* on page 409
- *Storage Size* on page 397

Date and Time Functions

Date and time functions perform conversion, extraction, or manipulation operations on date and time data types and can return date and time information.

These tables list the date and time functions and their parameters.

Syntax 1

Date and Time Functions	Parameters
DATE	(*expression*)
DATECEILING	(*date-part*, *datetime-expr*, *[multiple-expr]*)
DATEFLOOR	(*date-part*, *datetime-expr*, *[multiple-expr]*)
DATEFORMAT	(*datetime-expr*, *string-expr*)
DATENAME	(*date-part*, *date-expr*)
DATEROUND	(*date-part*, *datetime-expr*, *[multiple-expr]*)
DATETIME	(*expression*)
DAY	(*date-expr*)
DAYNAME	(*date-expr*)
DAYS	(*date-expr*)
DAYS	(*date-expr*, *date-expr*)
DAYS	(*date-expr*, *integer-expr*)
DOW	(*date-expr*)
HOUR	(*datetime-expr*)
HOURS	(*datetime-expr*)
HOURS	(*datetime-expr*, *datetime-expr*)
HOURS	(*datetime-expr*, *integer-expr*)
ISDATE	(*string*)
MINUTE	(*datetime-expr*)
MINUTES	(*datetime-expr*)
MINUTES	(*datetime-expr*, *datetime-expr*)

Date and Time Functions	**Parameters**
MINUTES	(*datetime-expr*, *integer-expr*)
MONTH	(*date-expr*)
MONTHNAME	(*date-expr*)
MONTHS	(*date-expr*)
MONTHS	(*date-expr*, *date-expr*)
MONTHS	(*date-expr*, *integer-expr*)
NOW	(*)
QUARTER	(*date-expr*)
SECOND	(*datetime-expr*)
SECONDS	(*datetime-expr*)
SECONDS	(*datetime-expr*, *datetime-expr*)
SECONDS	(*datetime-expr*, *integer-expr*)
TODAY	(*)
WEEKS	(*date-expr*)
WEEKS	(*date-expr*, *date-expr*)
WEEKS	(*date-expr*, *integer-expr*)
YEAR	(*date-expr*)
YEARS	(*date-expr*)
YEARS	(*date-expr*, *date-expr*)
YEARS	(*date-expr*, *integer-expr*)
YMD	(*year-num*, *month-num*, *day-num*)

Syntax 2

Transact-SQL Compatible Date and Time Functions	**Parameters**
DATEADD	(*date-part, numeric-expression, date-expr*)
DATEDIFF	(*date-part*, *date-expr1*, *date-expr2*)
DATENAME	(*date-part*, *date-expr*)

Transact-SQL Compatible Date and Time Functions	Parameters
DATEPART	(*date-part, date-expr*)
GETDATE	()

Description

SAP Sybase IQ provides two classes of date and time functions that can be used interchangeably, but have different styles. One set is Transact-SQL-compatible.

The date and time functions listed for Syntax 1 allow manipulation of time units. Most time units (such as `MONTH`) have four functions for time manipulation, although only two names are used (such as `MONTH` and `MONTHS`).

The functions listed for Syntax 2 are the Transact-SQL date and time functions. They allow an alternative way of accessing and manipulating date and time functions.

You should convert arguments to date functions to dates before used them. For example, this is incorrect:

```
days ( '1995-11-17', 2 )
```

This is correct:

```
days ( date( '1995-11-17' ), 2 )
```

SAP Sybase IQ does not have the same constants or data type promotions as SQL Anywhere, with which it shares a common user interface. If you issue a **SELECT** statement without a `FROM` clause, the statement is passed to SQL Anywhere. The following statement is handled exclusively by SQL Anywhere:

```
SELECT WEEKS(‘1998/11/01’);
```

The following statement, processed by SAP Sybase IQ, uses a different starting point for the **WEEKS** function and returns a different result than the statement above:

```
SELECT WEEKS(‘1998/11/01’) FROM iq_dummy;
```

Consider another example. The **MONTHS** function returns the number of months since an “arbitrary starting date.” The “arbitrary starting date” of SAP Sybase IQ, the imaginary date 0000-01-01, is chosen to produce the most efficient date calculations and is consistent across various data parts. SQL Anywhere does not have a single starting date. The following statements, the first processed by SQL Anywhere, the second by SAP Sybase IQ, both return the answer 12:

```
SELECT MONTHS('0001/01/01');
```

```
SELECT MONTHS('0001/01/01') FROM iq_dummy;
```

However, also consider these statements:

```
SELECT DAYS('0001/01/01');
SELECT DAYS('0001/01/01') FROM iq_dummy;
```

The first, processed by SQL Anywhere, yields the value 307, but the second, processed by SAP Sybase IQ, yields 166.

For the most consistent results, therefore, always include the table name in the **FROM** clause whether you need it or not.

Note: Create a dummy table with only one column and row. You can then reference this table in the **FROM** clause for any **SELECT** statement that uses date or time functions, thus ensuring processing by SAP Sybase IQ, and consistent results.

Date Parts

Many of the date functions use dates built from date parts.

This table displays allowed values of *date-part*.

Date Part	Abbreviation	Values
Year	yy	0001 – 9999
Quarter	qq	1 – 4
Month	mm	1 – 12
Week	wk	1 – 54
Day	dd	1 – 31
Dayofyear	dy	1 – 366
Weekday	dw	1 – 7 (Sun. – Sat.)
Hour	hh	0 – 23
Minute	mi	0 – 59
Second	ss	0 – 59
Millisecond	ms	0 – 999
Microsecond	mcs or us	0 – 999999
Calyearofweek	cyr	Integer. The year in which the week begins. The week containing the first few days of the year can be part of the last week of the previous year, depending upon which day it begins. If the new year starts on a Thursday through Saturday, its first week starts on the last Sunday of the previous year. If the new year starts on a Sunday through Wednesday, none of its days are part of the previous year.
Calweekofyear	cwk	An integer from 1 to 54 representing the week number within the year that contains the specified date.

Date Part	Abbreviation	Values
Caldayofweek	cdw	The day number within the week (Sunday = 1, Saturday = 7).

Note: By default, Sunday is the first day of the week. To make Monday be the first day, use:

```
set option ‘Date_First_Day_Of_Week’ = ‘1’
```

Compatibility

For compatibility with Adaptive Server Enterprise, use the Transact-SQL date and time functions.

See also

- *DATEADD Function [Date and Time]* on page 191
- *DATECEILING Function [Date and Time]* on page 192
- *DATEDIFF Function [Date and Time]* on page 195
- *DATEFLOOR Function [Date and Time]* on page 197
- *DATEPART Function [Date and Time]* on page 203
- *DATENAME Function [Date and Time]* on page 201
- *DATEROUND Function [Date and Time]* on page 204

HTTP Functions

HTTP functions facilitate the handling of HTTP requests within Web services.

Note: Ensure your Web services use best coding practices to safeguard against cross-site scripting (XSS) attacks. Open-source resources are available at organizations such as *OWASP*.

Table 10. HTTP functions

HTTP function	Parameters
HTML_DECODE	(*string*)
HTML_ENCODE	(*string*)
HTTP_DECODE	(*string*)
HTTP_ENCODE	(*string*)
HTTP_HEADER	(*header-field-name*)
HTTP_VARIABLE	(*var-name* [[, *instance*], *header-field*])

HTTP function	Parameters
NEXT_HTTP_HEADER	(*header-name*)
NEXT_HTTP_VARIABLE	(*var-name*)

Numeric Functions

Numeric functions perform mathematical operations on numerical data types or return numeric information.

Function

SAP Sybase IQ does not have the same constants or data type promotions as SQL Anywhere, with which it shares a common user interface. If you issue a **SELECT** statement without a **FROM** clause, the statement is passed through to SQL Anywhere. For the most consistent results, include the table name in the **FROM** clause whether you need it or not.

Note: Consider creating a dummy table to use in such cases.

This table lists numeric functions and their parameters.

Numeric Function	Parameters
ABS	(*numeric-expr*)
ACOS	(*numeric-expr*)
ASIN	(*numeric-expr*)
ATAN	(*numeric-expr*)
ATAN2	(*numeric-expr1*, *numeric-expr2*)
CEIL	(*numeric-expr*)
CEILING	(*numeric-expr*)
COS	(*numeric-expr*)
COT	(*numeric-expr*)
DEGREES	(*numeric-expr*)
EXP	(*numeric-expr*)
FLOOR	(*numeric-expr*)
LN	(*numeric-expr*)
LOG	(*numeric-expr*)

Numeric Function	Parameters
LOG10	(*numeric-expr*)
MOD	(*dividend, divisor*)
PI	(*)
POWER	(*numeric-expr1, numeric-expr2*)
RADIANS	(*numeric-expr*)
RAND	([*integer-expr*])
REMAINDER	(*numeric-expr, numeric-expr*)
ROUND	(*numeric-expr, integer-expr*)
SIGN	(*numeric-expr*)
SIN	(*numeric-expr*)
SQRT	(*numeric-expr*)
SQUARE	(*numeric-expr*)
TAN	(*numeric-expr*)
"TRUNCATE"	(*numeric-expr, integer-expr*)
TRUNCNUM	(*numeric-expression, integer-expression*)
WIDTH_BUCKET	(*expression, min_value, max_value, num_buckets*)

String Functions

String functions perform conversion, extraction, or manipulation operations on strings, or return information about strings.

When working in a multibyte character set, check carefully whether the function being used returns information concerning characters or bytes.

Most of the string functions accept binary data (hexadecimal strings) in the *string-expr* parameter, but some of the functions, such as **LCASE**, **UCASE**, **LOWER**, and **LTRIM**, expect the string expression to be a character string.

Unless you supply a constant **LENGTH** argument to a function that produces a `LONG VARCHAR` result (such as **SPACE** or **REPEAT**), the default length is the maximum allowed.

SAP Sybase IQ queries containing one or more of these functions might return one of the following errors:

```
ASA Error -1009080: Key doesn't fit on a single database page:
65560(4, 1)
```

```
ASA Error -1009119: Record size too large for database page size
```

For example:

```
SELECT COUNT(*) FROM test1 a WHERE (a.col1 + SPACE(4-LENGTH(a.col1))
+ a.col2 + space(2- LENGTH(a.col2))) IN (SELECT (b.col3) FROM test1
b);
```

To avoid such errors, cast the function result with an appropriate maximum length; for example:

```
SELECT COUNT(*) FROM test1 a WHERE (a.col1 + CAST(SPACE(4-
LENGTH(a.col1)) AS VARCHAR(4)) + a.col2 + CAST(SPACE(2-LENGTH
(a.col2)) AS VARCHAR(4))) IN (SELECT (b.col3) FROM test1 b);
```

The errors are more likely with a page size of 64K or a multibyte collation.

This table lists string functions and their parameters.

Table 11. String functions

String Function	Parameters
ASCII	(*string-expr*)
BIT_LENGTH	(*column-name*)
BYTE_LENGTH	(*string-expr*)
CHAR	(*integer-expr*)
CHAR_LENGTH	(*string-expr*)
CHARINDEX	(*string-expr1*, *string-expr2*)
DIFFERENCE	(*string-expr1*, *string-expr2*)
GRAPHICAL_PLAN	(*string-expr*)
HTML_PLAN	(*string-expr*)
INSERTSTR	(*numeric-expr*, *string-expr1*, *string-expr2*)
LCASE	(*string-expr*)
LEFT	(*string-expr*, *numeric-expr*)
LEN	(*string-expr*)
LENGTH	(*string-expr*)
LOCATE	(*string-expr1*, *string-expr2* [, *numeric-expr*])
LOWER	(*string-expr*)

String Function	Parameters
LTRIM	(*string-expr*)
OCTET_LENGTH	(*column-name*)
PATINDEX	('%*pattern*%', *string_expr*)
REPEAT	(*string-expr*, *numeric-expr*)
REPLACE	(*original-string*, *search-string*, *replace-string*)
REVERSE	(*expression* | *uchar_expr*)
REPLICATE	(*string-expr*, *integer-expr*)
RIGHT	(*string-expr*, *numeric-expr*)
RTRIM	(*string-expr*)
SIMILAR	(*string-expr1*, *string-expr2*)
SORTKEY	(*string-expression* [, { *collation-id* | *collation-name* [(*collation-tailoring-string*)] }])
SOUNDEX	(*string-expr*)
SPACE	(*integer-expr*)
STR	(*numeric_expr* [, *length* [, *decimal*]])
STR_REPLACE	(*string_expr1*, *string_expr2*, *string_expr3*)
STRING	(*string1* [, *string2*, …, *string99*])
STUFF	(*string-expr1*, *start*, *length*, *string-expr2*)
SUBSTRING	(*string-expr*, *integer-expr* [, *integer-expr*])
TRIM	(*string-expr*)
UCASE	(*string-expr*)
UPPER	(*string-expr*)

For information on string functions that support the `LONG BINARY` and `LONG VARCHAR` data types, see *Function Support* in *Unstructured Data Analytics*.

See also

- *Physical Limitations* on page 419

System Functions

System functions return system information.

This table lists the system functions and their parameters.

Table 12. System functions

System function	Parameters
COL_LENGTH	(*table-name*, *column-name*)
COL_NAME	(*table-id*, *column-id* [, *database-id*])
CONNECTION_PROPERTY	({ *property-id* | *property-name* } [, *connection-id*])
DATALENGTH	(*expression*)
DB_ID	([*database-name*])
DB_NAME	([*database-id*])
DB_PROPERTY	({ *property-id* | *property-name* } [,{*database-id* | *database-name* }])
EVENT_CONDITION	(*condition-name*)
EVENT_CONDITION_NAME	(*integer*)
EVENT_PARAMETER	(*context-name*)
GROUP_MEMBER	(*group-name-string-expression* [, *user-name-string-expression*])
INDEX_COL	(*table-name*, *index-id*, *key_#* [,*user-id*])
NEXT_CONNECTION	(*[connection-id] [, database-id]*)
NEXT_DATABASE	({ **NULL** | *database-id* })
OBJECT_ID	(*object-name*)
OBJECT_NAME	(*object-id* [, *database-id*])
PROPERTY	({ *property-id* | *property-name* })
PROPERTY_DESCRIPTION	(*property-id* | *property-name* })
PROPERTY_NAME	(*property-id*)
PROPERTY_NUMBER	(*property-name*)
SUSER_ID	([*user-name*])

System function	Parameters
SUSER_NAME	([*user-id*])
USER_ID	([*user-name*])
USER_NAME	([*user-id*])

Description

Databases currently running on a server are identified by a database name and a database ID number. The `db_id` and `db_name` functions provide information on these values.

A set of system functions provides information about properties of a currently running database, or of a connection, on the database server. These system functions take the database name or ID, or the connection name, as an optional argument to identify the database or connection for which the property is requested.

Performance

System functions are processed differently than other SAP Sybase IQ functions. When queries to SAP Sybase IQ tables include system functions, performance is reduced.

Compatibility

This table shows the Adaptive Server Enterprise system functions and their status in SAP Sybase IQ:

Table 13. Status of Adaptive Server Enterprise system functions in SAP Sybase IQ

Function	Status
col_length	Implemented
col_name	Implemented
db_id	Implemented
db_name	Implemented
index_col	Implemented
object_id	Implemented
object_name	Implemented
proc_role	Always returns 0
show_role	Always returns NULL
tsequal	Not implemented

Function	Status
user_id	Implemented
user_name	Implemented
suser_id	Implemented
suser_name	Implemented
datalength	Implemented
curunreservedpgs	Not implemented
data_pgs	Not implemented
host_id	Not implemented
host_name	Not implemented
lct_admin	Not implemented
reserved_pgs	Not implemented
rowcnt	Not implemented
used_pgs	Not implemented
valid_name	Not implemented
valid_user	Not implemented

Notes

- Some of the system functions are implemented in SAP Sybase IQ as system stored procedures.
- The `db_id`, `db_name`, `datalength`, `suser_id`, and `suser_name` functions are implemented as built-in functions.

Connection Properties

Retrieve the value of a specific connection property or the values of all connection properties.

Examples

Use the `connection_property` system function to retrieve the value of a connection property. The following statement returns the number of pages that have been read from file by the current connection:

```
select connection_property ( 'DiskRead' )
```

Use the `sa_conn_properties` system procedure to retrieve the values of all connection properties.

```
call sa_conn_properties
```

A separate row appears for each connection, for each property.

See also

- *PROPERTY Function [System]* on page 301
- *PROPERTY_NAME Function [System]* on page 303
- *PROPERTY_NUMBER Function [System]* on page 303
- *CONNECTION_PROPERTY Function [System]* on page 176

Properties Available for the Server

Retrieve the value of a specific server property or the values of all server properties.

Server properties apply across the server as a whole.

The Server Edition property returns the SQL Anywhere edition, not the SAP Sybase IQ edition. To show SQL Anywhere license information, use the **sp_iqlmconfig** system procedure.

Examples

Use the `property` system function to retrieve the value of a server property. The following statement returns the number of cache pages being used to hold the main heap:

```
select property ( 'MainHeapPages') from iq_dummy
```

Use the `sa_eng_properties` system procedure to retrieve the values of all server properties.

```
call sa_eng_properties
```

See also

- *sp_iqlmconfig Procedure* on page 513
- *PROPERTY Function [System]* on page 301
- *PROPERTY_NAME Function [System]* on page 303
- *PROPERTY_NUMBER Function [System]* on page 303
- *CONNECTION_PROPERTY Function [System]* on page 176

List of connection properties

Connection properties are available for each connection to a SAP Sybase IQ database. Connection property names are case insensitive.

Connection properties

Property name	Description
`allow_nulls_by_default`	Returns a value indicating whether columns created without specifying either NULL or NOT NULL are allowed to contain NULL values.
`allow_read_client_file`	Returns a value indicating whether the database server allows the reading of files on a client computer.
`allow_snapshot_isolation`	Returns a value indicating whether snapshot isolation is enabled or disabled.
`allow_write_client_file`	Returns a value indicating whether the database server allows the writing of files to a client computer.
`ansi_blanks`	Returns a value indicating when character data is truncated at the client side.
`ansi_close_cursors_on_rollback`	Returns a value indicating whether cursors opened WITH HOLD are closed when a ROLLBACK is performed.
`ansi_permissions`	Returns a value indicating whether privileges are checked for DELETE and UPDATE statements.
`ansi_substring`	Returns a value indicating how the SUBSTRING (SUBSTR) function behaves when negative values are provided for the start or length parameters.
`ansi_update_constraints`	Returns a value indicating the range of updates that are permitted.
`ansinull`	Returns a value that indicates how NULL values are interpreted.

Property name	Description
AppInfo	Returns information about the client that made the connection. For HTTP connections, this includes information about the browser. For connections using older versions of jConnect or Sybase Open Client, the information may be incomplete. The API value can be DBLIB, ODBC, OLEDB, ADO.NET, iAnywhereJDBC, PHP, PerlDBD, or DBEXPRESS.
ApproximateCPUTime	Returns an estimate of the amount of CPU time accumulated by a given connection, in seconds. The value returned may differ from the actual value by as much as 50%, although typical variations are in the 5-10% range. On multi-processor computers, each CPU (or hyperthread or core) accumulates time, so the sum of accumulated times for all connections may be greater than the elapsed time. This property is supported on Windows and Linux.
auditing	Returns On if the PUBLIC.auditing option is set to On. Otherwise, returns Off. If the auditing option is set to On, and the conn_auditing option is set to Off, the auditing connection property still returns On, even though the current connection is not being audited.
auditing_options	This property is reserved for system use. Do not change the setting of this option.
Authenticated	Returns Yes if the application has sent a valid connection authentication string. Returns No if the application has not sent a valid connection authentication string.
AuthType	Returns the type of authentication used when connecting. The value returned is one of Standard, Integrated, Kerberos, LDAPUA, or an empty string. An empty string is returned for internal connections and connections for HTTP services that use AUTHORIZATION OFF.

Property name	Description
`auto_commit_on_create_lo-cal_temp_index`	Returns the setting of the auto_commit_on_cre-ate_local_temp_index option.
`background_priority`	This property is deprecated. Returns a value in-dicating how much impact the current connection has on the performance of other connections.
`BlockedOn`	Returns zero if the current connection isn't blocked, or if it is blocked, the connection number on which the connection is blocked because of a locking conflict.
`blocking`	Returns a value indicating the database server's behavior in response to locking conflicts.
`blocking_others_timeout`	Returns the length of time that another connec-tion can block on the current connection's row and table locks before the current connection is rolled back.
`blocking_timeout`	Returns the length of time, in milliseconds, a transaction waits to obtain a lock.
`BytesReceived`	Returns the number of bytes received during cli-ent/server communications. This value is updated for HTTP and HTTPS connections.
`BytesReceivedUncomp`	Returns the number of bytes that would have been received during client/server communications if compression was disabled. This value is the same as the value for BytesReceived if compression is disabled.
`BytesSent`	Returns the number of bytes sent during client/ server communications. This value is updated for HTTP and HTTPS connections.
`BytesSentUncomp`	Returns the number of bytes that would have been sent during client/server communications if com-pression was disabled. This value is the same as the value for BytesSent if compression is disa-bled.
`CacheHits`	Returns the number of successful reads of the cache.

Property name	Description
`CacheRead`	Returns the number of database pages that have been looked up in the cache.
`CacheReadIndInt`	Returns the number of index internal-node pages that have been read from the cache.
`CacheReadIndLeaf`	Returns the number of index leaf pages that have been read from the cache.
`CacheReadTable`	Returns the number of table pages that have been read from the cache.
`CacheReadWorkTable`	Returns the number of cache work table reads.
`CarverHeapPages`	Returns the number of heap pages used for short-term purposes such as query optimization.
`chained`	Returns the transaction mode used in the absence of a BEGIN TRANSACTION statement.
`CharSet`	Returns the CHAR character set used by the connection. This property has extensions that you can specify when querying the property value.
`checkpoint_time`	Returns the maximum time, in minutes, that the database server runs without doing a checkpoint.
`cis_option`	Returns 7 if debugging information for remote data access appears in the database server messages window and 0 if the debugging information for remote data access does not appear in the database server messages window.
`cis_rowset_size`	Returns the number of rows that are returned from remote servers for each fetch.
`ClientLibrary`	Returns jConnect for jConnect connections; CT_Library for Sybase Open Client connections; None for HTTP connections, and CmdSeq for ODBC, embedded SQL, OLE DB, ADO.NET, and SAP Sybase IQ JDBC driver connections.

Property name	Description
`ClientNodeAddress`	Returns the node for the client in a client/server connection. When the client and server are both on the same computer, an empty string is returned. This is a synonym for the NodeAddress property. This property returns `NA` if the request that is currently executing is part of an event handler.
`ClientPort`	Returns the client's TCP/IP port number or 0 if the connection isn't a TCP/IP connection.
`ClientStmtCacheHits`	Returns the number of prepares that were not required for this connection because of the client statement cache. This is the number of additional prepares that would be required if client statement caching was disabled.
`ClientStmtCacheMisses`	Returns the number of statements in the client statement cache for this connection that were prepared again. This is the number of times a cached statement was considered for reuse, but could not be reused because of a schema change, a database option setting, or a DROP VARIABLE statement.
`close_on_endtrans`	Returns On or Off to indicate whether cursors are closed at the end of a transaction.
`collect_statistics_on_dml_updates`	Returns On or Off to indicate whether statistics are gathered during the execution of data-altering DML statements such as INSERT, DELETE, and UPDATE.
`Commit`	Returns the number of Commit requests that have been handled.
`CommLink`	Returns the communication link for the connection. This is one of the network protocols supported by SAP Sybase IQ, or local for a same-computer connection. This property returns `NA` if the request that is currently executing is part of an event handler.

Property name	Description
`CommNetworkLink`	Returns the communication link for the connection. This is one of the network protocols supported by SAP Sybase IQ. Values include SharedMemory and TCPIP. The CommNetworkLink property always returns the name of the link, regardless of whether it is same-computer or not. This property returns `NA` if the request that is currently executing is part of an event handler.
`CommProtocol`	Returns TDS for Sybase Open Client and jConnect connections, HTTP for HTTP connections, HTTPS for HTTPS connections, and CmdSeq for ODBC, embedded SQL, OLE DB, ADO.NET, and SAP Sybase IQ JDBC driver connections.
`Compression`	Returns On or Off to indicate whether communication compression is enabled on the connection. This property returns `NA` if the request that is currently executing is part of an event handler.
`conn_auditing`	Returns On if auditing is enabled for the connection, even if the auditing option is set to Off.
`ConnectedTime`	Returns the total length of time, in seconds, that a connection has been connected.
`connection_authentication`	Returns the string used to authenticate the client. Authentication is required before the database can be modified.
`continue_after_raiserror`	Returns On or Off to indicate whether execution of a procedure or trigger is stopped whenever the RAISERROR statement is encountered.
`conversion_error`	Returns On or Off to indicate data type conversion failures are reported when fetching information from the database.
`cooperative_commit_timeout`	Returns the time, in milliseconds, that the database server waits for other connections to fill a page of the log before writing to disk.
`cooperative_commits`	Returns On or Off to indicate when commits are written to disk.

Property name	Description
`CurrentLineNumber`	Returns the current line number of the procedure or compound statement a connection is executing. The procedure can be identified using the CurrentProcedure property. If the line is part of a compound statement from the client, an empty string is returned.
`CurrentProcedure`	Returns the name of the procedure that a connection is currently executing. If the connection is executing nested procedure calls, the name is the name of the current procedure. If there is no procedure executing, an empty string is returned.
`Cursor`	Returns the number of declared cursors that are currently being maintained by the database server.
`CursorOpen`	Returns the number of open cursors that are currently being maintained by the database server.
`database_authentication`	Returns the string used to authenticate the database. Authentication is required for authenticated database servers before the database can be modified.
`date_format`	Returns a string indicating the format for dates retrieved from the database.
`date_order`	Returns a string indicating how dates are formatted.
`db_publisher`	Returns the user ID of the database publisher.
`DBNumber`	Returns the ID number of the database.
`debug_messages`	Returns On or Off to indicate whether MESSAGE statements that include a DEBUG ONLY clause are executed.
`dedicated_task`	Returns On or Off to indicate whether a request handling task is dedicated exclusively to handling requests for the connection.
`default_dbspace`	Returns the name of the default dbspace, or an empty string if the default dbspace has not been specified.

Property name	Description
`default_timestamp_increment`	Returns a value, in microseconds, that is added to a column of type TIMESTAMP to keep values in the column unique.
`delayed_commit_timeout`	Returns the time, in milliseconds, that the database server waits to return control to an application following a COMMIT.
`delayed_commits`	Returns On or Off to indicate when the database server returns control to an application following a COMMIT.
`disk_sandbox`	Returns On or Off to indicate whether the read-write file operations of the database are restricted to the directory where the main database file is located.
`DiskRead`	Returns the number of pages that have been read from disk.
`DiskReadHint`	Returns the number of disk read hints.
`DiskReadHintPages`	Returns the number of disk read hint pages.
`DiskReadIndInt`	Returns the number of index internal-node pages that have been read from disk.
`DiskReadIndLeaf`	Returns the number of index leaf pages that have been read from disk.
`DiskReadTable`	Returns the number of table pages that have been read from disk.
`DiskReadWorkTable`	Returns the number of disk work table reads.
`DiskSyncRead`	Returns the number of disk reads issued synchronously.
`DiskSyncWrite`	Returns the number of writes issued synchronously.
`DiskWaitRead`	Returns the number of times the database server waited for an asynchronous read.
`DiskWaitWrite`	Returns the number of times the database server waited for an asynchronous write.

Property name	Description
`DiskWrite`	Returns the number of modified pages that have been written to disk.
`DiskWriteHint`	Returns the number of disk write hints.
`DiskWriteHintPages`	Returns the number of disk write hint pages.
`divide_by_zero_error`	Returns On if division by zero results in an error and Off if division by zero is not an error.
`Encryption`	Returns a value that indicates whether the connection is encrypted.
`escape_character`	This property is reserved for system use. Do not change the setting of this option.
`EventName`	Returns the name of the associated event if the connection is running an event handler. Otherwise, an empty string is returned.
`exclude_operators`	This property is reserved for system use. Do not change the setting of this option.
`ExprCacheAbandons`	Returns the number of times that the expression cache was abandoned because the hit rate was too low.
`ExprCacheDropsToReadOnly`	Returns the number of times that the expression cache dropped to read-only status because the hit rate was low.
`ExprCacheEvicts`	Returns the number of evictions from the expression cache.
`ExprCacheHits`	Returns the number of hits in the expression cache.
`ExprCacheInserts`	Returns the number of values inserted into the expression cache.
`ExprCacheLookups`	Returns the number of lookups done in the expression cache.
`ExprCacheResumesOfReadWrite`	Returns the number of times that the expression cache resumed read-write status because the hit rate increased.

Property name	Description
ExprCacheStarts	Returns the number of times that the expression cache was started.
extended_join_syntax	Returns On if queries with duplicate correlation name syntax for multi-table joins are allowed, Off if they are reported as an error.
extern_login_credentials	Returns whether the remote connections are attempted using the logged in user's external login credentials or the effective user's external login credentials.
fire_triggers	Returns On if triggers are fired in the database, otherwise, returns Off.
first_day_of_week	Returns the number that is used for the first day of the week, where 7=Sunday and 1=Monday.
for_xml_null_treatment	Returns Omit if elements and attributes that contain NULL values are omitted from the result and Empty if empty elements or attributes are generated for NULL values when the FOR XML clause is used in a query.
force_view_creation	This property is reserved for system use. Do not change the setting of this option.
FullCompare	Returns the number of comparisons that have been performed beyond the hash value in an index.
GetData	Returns the number of GETDATA requests.
global_database_id	Returns the starting value used for columns created with DEFAULT GLOBAL AUTOINCREMENT.
HashForcedPartitions	Returns the number of times that a hash operator was forced to partition because of competition for memory.
HashRowsFiltered	Returns the number of probe rows rejected by bit-vector filters.
HashRowsPartitioned	Returns the number of rows written to hash work tables.

Property name	Description
`HashWorkTables`	Returns the number of work tables created for hash-based operations.
`HeapsCarver`	Returns the number of heaps used for short-term purposes such as query optimization.
`HeapsLocked`	Returns number of relocatable heaps currently locked in the cache.
`HeapsQuery`	Returns the number of heaps used for query processing (hash and sort operations).
`HeapsRelocatable`	Returns the number of relocatable heaps.
`http_connection_pool_basesize`	Returns the nominal threshold size of database connections.
`http_connection_pool_timeout`	Returns the maximum length of time that unused connections are stored in the connection pool.
`http_session_timeout`	Returns the current HTTP session timeout, in minutes.
`HttpServiceName`	Returns the service name entry point for the current HTTP request. This property is useful for error reporting and control flow. An empty string is returned when this property is selected from a stored procedure that did not originate from an HTTP request or if the connection is currently inactive or waiting to continue an HTTP session.
`IdleTimeout`	Returns the idle timeout value of the connection. This property returns `NA` if the request that is currently executing is part of an event handler.
`IndAdd`	Returns the number of entries that have been added to indexes.
`IndLookup`	Returns the number of entries that have been looked up in indexes.
`integrated_server_name`	Returns the name of the Domain Controller server used for looking up Windows user group membership for integrated logins.

Property name	Description
IsDebugger	Returns Yes or No to distinguish connections that are being used to run the SAP Sybase IQ debugger. The value is Yes if the current connection number corresponds to the connection number of a debugger connection, and No otherwise.
isolation_level	Returns the isolation level of the connection (0, 1, 2, 3, Snapshot, Statement-snapshot, or Readonly-statement-snapshot).
java_class_path	Returns a list of additional directories or JAR files that are searched for classes.
java_location	Returns the path of the Java VM for the database if one has been specified.
java_main_userid	This property is deprecated.
java_vm_options	Returns the command line options that the database server uses when it launches the Java VM.
Language	Returns the locale language.
LastCommitRedoPos	Returns the redo log position after the last COMMIT operation was written to the transaction log by the connection.
LastIdle	Returns the number of ticks between requests.
LastPlanText	Returns the long text plan of the last query executed on the connection. You control the remembering of the last plan by setting the RememberLastPlan option of the sa_server_option system procedure, or using the -zp server option.
LastReqTime	Returns the time at which the last request for the specified connection started. This property can return an empty string for internal connections, such as events.

Property name	Description
`LastStatement`	Returns the most recently prepared SQL statement for the current connection. The LastStatement value is set when a statement is prepared, and is cleared when a statement is dropped. Only one statement string is remembered for each connection. If sa_conn_activity reports a non-empty value for a connection, it is most likely the statement that the connection is currently executing. If the statement had completed, it would likely have been dropped and the property value would have been cleared. If an application prepares multiple statements and retains their statement handles, then the LastStatement value does not reflect what a connection is currently doing. When client statement caching is enabled, and a cached statement is reused, this property returns an empty string.
`LivenessTimeout`	Returns the liveness timeout period for the current connection. This property returns `NA` if the request that is currently executing is part of an event handler.
`lock_rejected_rows`	This property is reserved for system use. Do not change the setting of this option.
`LockCount`	Returns the number of locks held by the connection.
`LockIndexID`	Returns the identifier of the locked index.
`LockName`	Returns a 64-bit unsigned integer value representing the lock for which a connection is waiting.
`LockRowID`	Returns the identifier of the locked row.

Property name	Description
`LockTableOID`	Returns zero if the connection isn't blocked, or if the connection is on a different database than the connection calling CONNECTION_PROPERTY. Otherwise, this is the object ID of the table for the lock on which this connection is waiting. The object ID can be used to look up table information using the SYSTAB system view.
`log_deadlocks`	Returns On if deadlock information is reported; otherwise, returns Off.
`LogFreeCommit`	Returns the number of redo free commits. A redo free commit occurs when a commit of the transaction log is requested but the log has already been written (so the commit was done for free.)
`login_mode`	Returns one or more of Standard, Integrated, Kerberos, or LDAPUA to indicate the type of logins that are supported.
`login_procedure`	Returns the name of the stored procedure used to set compatibility options at startup.
`LoginTime`	Returns the date and time the connection was established.
`LogWrite`	Returns the number of pages that have been written to the transaction log.
`materialized_view_optimization`	Returns a value indicating whether materialized views are used during query optimization: Disabled Fresh Stale *N* Minute[s] *N* Hour[s] *N* Day[s] *N* Week[s] *N* Month[s]
`max_client_statements_cached`	Returns the number of statements cached by the client.

Property name	Description
`max_cursor_count`	Returns a value specifying the maximum number of cursors that a connection can use at once.
`max_hash_size`	This property is deprecated.
`max_plans_cached`	Returns a value specifying the maximum number of execution plans to be stored in a cache.
`max_priority`	Returns a value indicating the maximum priority level a connection can have.
`max_query_tasks`	Returns the maximum number of requests that the database server can use to process a query.
`max_recursive_iterations`	Returns a value specifying the maximum number of iterations a recursive common table expression can make.
`max_statement_count`	Returns a value specifying the maximum number of prepared statements that a connection can use simultaneously.
`max_temp_space`	Returns a value indicating the maximum amount of temporary file space available for a connection.
`MessageReceived`	Returns the string that was generated by the MESSAGE statement that caused the WAITFOR statement to be interrupted. Otherwise, an empty string is returned.
`min_password_length`	Returns the minimum length for new passwords in the database.
`min_role_admins`	Returns the minimum number of administrators required for a role. See min_role_admins option.

Property name	Description
`Name`	Returns the name of the current connection. You can specify a connection name using the ConnectionName (CON) connection parameter. The following names are used for temporary connections created by the database server: INT:ApplyRecovery INT:BackupDB INT:Checkpoint INT:Cleaner INT:CloseDB INT:CreateDB INT:CreateMirror INT:DelayedCommit INT:DiagRcvr INT:DropDB INT:EncryptDB INT:Exchange INT:FlushMirrorLog INT:FlushStats INT:HTTPReq INT:PromoteMirror INT:PurgeSnapshot INT:ReconnectMirror INT:RecoverMirror INT:RedoCheckpoint INT:RefreshIndex INT:ReloadTrigger INT:RenameMirror INT:RestoreDB INT:StartDB INT:VSS
`NcharCharSet`	Returns the NCHAR character set used by the connection. This property has extensions that you can specify when querying the property value.

Property name	Description
`nearest_century`	Returns a value that indicates how two-digit years are interpreted in string-to-date conversions.
`NodeAddress`	Returns the node for the client in a client/server connection. When the client and server are both on the same computer, an empty string is returned.
`non_keywords`	Returns a list of keywords, if any, that are turned off so they can be used as identifiers.
`Number`	Returns the connection ID (a number) for the current connection.
`NumLocalTempTables`	Returns the number of local temporary tables in use by the connection. When a local temporary table is dropped or goes out of scope, it is still considered to be in use until the next COMMIT.
`odbc_describe_binary_as_varbinary`	Returns Off if the SAP Sybase IQ ODBC driver describes both BINARY and VARBINARY columns as SQL_BINARY and returns On if the ODBC driver describes BINARY and VARBINARY columns as SQL_VARBINARY.
`odbc_distinguish_char_and_varchar`	Returns Off if CHAR columns are described as SQL_VARCHAR, and On if CHAR columns are described as SQL_CHAR.
`oem_string`	Returns the string stored in the header page of the database file.
`on_charset_conversion_failure`	Returns one of Ignore, Warning, or Error to indicate the behavior when an error is encountered during character set conversion.
`on_tsql_error`	Returns one of Stop, Conditional, or Continue to indicate the behavior when an error is encountered while executing a stored procedure or T-SQL batch.
`optimization_goal`	Returns one of First-row or All-rows to indicate how query processing is optimized.

Property name	Description
`optimization_level`	Returns a value between 0 and 15. This number is used to control the level of effort made by the SAP Sybase IQ query optimizer to find an access plan for a SQL statement.
`optimization_workload`	Returns a value indicating the level of effort made by the SAP Sybase IQ query optimizer to find an access plan for a SQL statement.
`OSUser`	Returns the operating system user name associated with the client process. If the client process is impersonating another user (or the set ID bit is set on Unix), the impersonated user name is returned. An empty string is returned for version 10.0.1 and earlier clients, and for HTTP and TDS clients.
`PacketSize`	Returns the packet size used by the connection, in bytes. This property returns `NA` if the request that is currently executing is part of an event handler.
`PacketsReceived`	Returns the number of client/server communication packets received. This value is not updated for HTTP or HTTPS connections.
`PacketsReceivedUncomp`	Returns the number of packets that would have been received during client/server communications if compression was disabled. (This value is the same as the value for PacketsReceived if compression is disabled.)
`PacketsSent`	Returns the number of client/server communication packets sent. This value is not updated for HTTP or HTTPS connections.
`PacketsSentUncomp`	Returns the number of packets that would have been sent during client/server communications if compression was disabled. (This value is the same as the value for PacketsSent if compression is disabled.)

Property name	Description
ParentConnection	Returns the connection ID of the connection that created a temporary connection to perform a database operation (such as performing a backup or creating a database). For other types of connections, this property returns NULL.
pinned_cursor_percent_of_cache	Returns the percentage of the cache that can be used for pinning cursors.
post_login_procedure	Returns the name of the procedure whose result set contains messages that should be displayed by applications when a user connects.
precision	Returns the decimal and numeric precision setting.
prefetch	Returns Off if no prefetching is done, Conditional if prefetching occurs unless the cursor type is SENSITIVE or the query includes a proxy table, or Always if prefetching is done even for SENSITIVE cursor types and cursors that involve a proxy table.
Prepares	Returns the number of statement preparations performed for the connection.
PrepStmt	Returns the number of prepared statements currently being maintained by the database server.
preserve_source_format	Returns On if the original source definition of procedures, triggers, views, and event handlers is saved in system tables, otherwise, returns Off.
prevent_article_pkey_update	Returns On if updates are not allowed to the primary key columns of tables involved in publications, otherwise returns Off.
priority	Returns a value indicating the priority level of a connection.

Property name	Description
`Progress`	Returns information about how long a query has been running. For example: `43% (9728 of 22230 pages) complete after 00:00:05; estimated 00:00:06 remaining` This property has extensions that you can specify when querying the property value.
`progress_messages`	Returns the value of the progress_messages option.
`query_mem_timeout`	Returns the value of the query_mem_timeout option.
`QueryBypassed`	Returns the number of requests optimized by the optimizer bypass.
`QueryBypassedCosted`	Returns the number of requests processed by the optimizer bypass using costing.
`QueryBypassedHeuristic`	Returns the number of requests processed by the optimizer bypass using heuristics.
`QueryBypassedOptimized`	Returns the number of requests initially processed by the optimizer bypass and subsequently fully optimized by the SAP Sybase IQ optimizer.
`QueryCachedPlans`	Returns the number of query execution plans currently cached for the connection.
`QueryCachePages`	Returns the number of cache pages used to cache execution plans.
`QueryDescribedBypass`	Returns the number of describe requests processed by the optimizer bypass.
`QueryDescribedOptimizer`	Returns the number of describe requests processed by the optimizer.
`QueryHeapPages`	Returns the number of cache pages used for query processing (hash and sort operations).
`QueryJHToJNLOptUsed`	Returns the number of times a hash join was converted to a nested loops join.

Property name	Description
`QueryLowMemoryStrategy`	Returns the number of times the server changed its execution plan during execution as a result of low memory conditions. The strategy can change because less memory is currently available than the optimizer estimated, or because the execution plan required more memory than the optimizer estimated.
`QueryMemActiveCurr`	Returns the number of requests actively using query memory.
`QueryMemGrantFailed`	Returns the total number of times a request waited for query memory, but failed to get it.
`QueryMemGrantGranted`	Returns the number of pages currently granted to requests.
`QueryMemGrantRequested`	Returns the total number of times any request attempted to acquire query memory.
`QueryMemGrantWaited`	Returns the total number of times any request waited for query memory.
`QueryMemGrantWaiting`	Returns the current number of requests waiting for query memory.
`QueryOpened`	Returns the number of OPEN requests for execution.
`QueryOptimized`	Returns the number of requests fully optimized.
`QueryReused`	Returns the number of requests that have been reused from the plan cache.
`QueryRowsFetched`	Returns the number of rows that have been read from base tables, either by a sequential scan or an index scan, for this connection.
`QueryRowsMaterialized`	Returns the number of rows that are written to work tables during query processing.
`quoted_identifier`	Returns On if strings enclosed in double quotes are interpreted as identifiers, or Off if they are interpreted as literal strings.

Property name	Description
`read_past_deleted`	Returns On if sequential scans at isolation levels 1 and 2 skip uncommitted deleted rows, and Off if sequential scans block on uncommitted deleted rows at isolation levels 1 and 2.
`recovery_time`	Returns the maximum length of time, in minutes, that the database server will take to recover from system failure.
`RecursiveIterations`	Returns the number of iterations for recursive unions.
`RecursiveIterationsHash`	Returns the number of times recursive hash join used a hash strategy.
`RecursiveIterationsNested`	Returns the number of times recursive hash join used a nested loops strategy.
`RecursiveJNLMisses`	Returns the number of index probe cache misses for recursive hash join.
`RecursiveJNLProbes`	Returns the number of times recursive hash join attempted an index probe.
`remote_idle_timeout`	Returns the time, in seconds, of inactivity that web service client procedures and functions will tolerate.
`replicate_all`	For internal use only.
`ReqCountActive`	Returns the number of requests processed, or NULL if the RequestTiming server property is set to Off.
`ReqCountBlockContention`	Returns the number of times the connection wai-ted for atomic access, or NULL if the -zt option was not specified.
`ReqCountBlockIO`	Returns the number of times the connection wai-ted for I/O to complete, or NULL if the -zt option was not specified.
`ReqCountBlockLock`	Returns the number of times the connection wai-ted for a lock, or NULL if the -zt option was not specified.

Property name	Description
`ReqCountUnscheduled`	Returns the number of times the connection waited for scheduling, or NULL if the -zt option was not specified.
`ReqStatus`	Returns the status of the request. It can be one of the following values: • **Idle –** The connection is not currently processing a request. • **Unscheduled* –** The connection has work to do and is waiting for an available database server worker. • **BlockedIO* –** The connection is blocked waiting for an I/O. • **BlockedContention* –** The connection is blocked waiting for access to shared database server data structures. • **BlockedLock –** The connection is blocked waiting for a locked object. • **Executing –** The connection is executing a request. The values marked with an asterisk (*) are only returned when logging of request timing information has been turned on for the database server using the -zt server option. If request timing information is not being logged (the default), the values are reported as Executing.
`ReqTimeActive`	Returns the amount of time, in seconds, spent processing requests, or NULL if the -zt option was not specified.
`ReqTimeBlockContention`	Returns the amount of time, in seconds, spent waiting for atomic access, or NULL if the RequestTiming server property is set to Off.
`ReqTimeBlockIO`	Returns the amount of time, in seconds, spent waiting for I/O to complete, or NULL if the -zt option was not specified.
`ReqTimeBlockLock`	Returns the amount of time, in seconds, spent waiting for a lock, or NULL if the -zt option was not specified.

Property name	Description
ReqTimeUnscheduled	Returns the amount of unscheduled time, or NULL if the -zt option was not specified.
ReqType	Returns the type of the last request. If a connection has been cached by connection pooling, its ReqType value is CONNECT_POOL_CACHE.
request_timeout	Returns the maximum time a single request can run.
RequestsReceived	Returns the number of client/server communication requests or round trips. It is different from PacketsReceived in that multi-packet requests count as one request, and liveness packets are not included.
reserved_keywords	Returns a list of non-default reserved keywords that are enabled for the database.
return_date_time_as_string	Returns On if DATE, TIME, and TIMESTAMP values are returned to applications as a string, and Off if they are returned as a DATE or TIME data type.
Rlbk	The number of rollback requests that have been handled.
rollback_on_deadlock	Returns After when referential integrity actions are executed after the UPDATE or DELETE, and Before if they are executed before the UPDATE or DELETE.
RollbackLogPages	Returns the number of pages in the rollback log.
row_counts	Returns On if the row count is always accurate, and Off if the row count is usually an estimate.
scale	Returns the decimal and numeric scale for the connection.
secure_feature_key	Stores the key that is used to enable and disable features for a database server. Selecting the value of this property always returns an empty string.

Property name	Description
`ServerNodeAddress`	Returns the node for the server in a client/server connection. When the client and server are both on the same computer, an empty string is returned. This property returns `NA` if the request that is currently executing is part of an event handler.
`ServerPort`	Returns the database server's TCP/IP port number or 0.
`SessionCreateTime`	Returns the time the HTTP session was created.
`SessionID`	Returns the session ID for the connection if it exists, otherwise, returns an empty string.
`SessionLastTime`	Returns the time of the last request for the HTTP session.
`SessionTimeout`	Returns the time, in minutes, the HTTP session persists during inactivity.
`SnapshotCount`	Returns the number of snapshots associated with the connection.
`sort_collation`	Returns Internal if the ORDER BY clause remains unchanged, otherwise the collation name or collation ID is returned.
`SortMergePasses`	Returns the number of merge passes used during sorting.
`SortRowsMaterialized`	Returns the number of rows written to sort work tables.
`SortRunsWritten`	Returns the number of sorted runs written during sorting.
`SortSortedRuns`	Returns the number of sorted runs created during run formation.
`SortWorkTables`	Returns the number of work tables created for sorting.

Property name	Description
`sql_flagger_error_level`	Returns one of the following values to indicate which SQL that is not part of a specified set of SQL/2003 is flagged as an error: • **E –** Flag syntax that is not entry-level SQL/2003 syntax • **I –** Flag syntax that is not intermediate-level SQL/2003 syntax • **F –** Flag syntax that is not full-SQL/2003 syntax • **W** – Allow all supported syntax
`sql_flagger_warning_level`	Returns one of the following values to indicate which SQL that is not part of a specified set of SQL/2003 is flagged as a warning: • **E –** Flag syntax that is not entry-level SQL/2003 syntax • **I –** Flag syntax that is not intermediate-level SQL/2003 syntax • **F –** Flag syntax that is not full-SQL/2003 syntax • **W** – Allow all supported syntax
`st_geometry_asbinary_format`	Returns a value that indicates how spatial values are converted from a geometry to a binary format.
`st_geometry_astext_format`	Returns a value that indicates how spatial values are converted from a geometry to text.
`st_geometry_asxml_format`	Returns a value that indicates how spatial values are converted from a geometry to xml.
`st_geometry_describe_type`	Returns a value that indicates spatial values are described.
`st_geometry_interpolation`	Returns the interpolation setting for ST_CircularString geometries.
`st_geometry_on_invalid`	Returns a value that indicates the behavior when a geometry fails surface validation.
`StatementDescribes`	Returns the total number of statements processed by DESCRIBE requests.

Property name	Description
`StatementPostAnnotates`	Returns the number of statements processed by the semantic query transformation phase.
`StatementPostAnnotatesSimple`	Returns the number of statements processed by the semantic query transformation phase, but that skipped some of the semantic transformations.
`StatementPostAnnotatesSkipped`	Returns the number of statements that have completely skipped the semantic query transformation phase.
`string_rtruncation`	Returns On if an error is raised when a string is truncated, and returns Off if an error is not raised and the character string is silently truncated.
`subsume_row_locks`	Returns On if the database server acquires individual row locks for a table, otherwise, returns Off.
`suppress_tds_debugging`	Returns Off if TDS debugging information appears in the database server messages window, and returns On if debugging information does not appear in the database server messages window.
`synchronize_mirror_on_commit`	Returns On if the database mirror server is synchronized on commit, otherwise returns Off.
`tds_empty_string_is_null`	Returns On if empty strings are returned as NULL for TDS connections, and returns Off if a string containing one blank character is returned for TDS connections.
`temp_space_limit_check`	Returns On if the database server checks the amount of temporary space available for a connection, and returns Off if the database server does not check the amount of space available for a connection.
`TempFilePages`	Returns the number of temporary file pages used by the connection.
`TempTablePages`	Returns the number of pages in the temporary file used for temporary tables.
`time_format`	Returns the string format used for times retrieved from the database.

Property name	Description
`time_zone_adjustment`	Returns the number of minutes that must be added to the Coordinated Universal Time (UTC) to display time local to the connection.
`timestamp_format`	Returns the format for timestamps that are retrieved from the database.
`timestamp_with_time_zone_format`	Returns the format for TIMESTAMP WITH TIME ZONE values retrieved from the database.
`TimeZoneAdjustment`	Returns the number of minutes that must be added to the Coordinated Universal Time (UTC) to display time local to the connection.
`TransactionStartTime`	Returns a string containing the time the database was first modified after a COMMIT or ROLLBACK, or an empty string if no modifications have been made to the database since the last COMMIT or ROLLBACK.
`truncate_timestamp_values`	Returns On if the number of decimal places used in the TIMESTAMP values is limited, otherwise, returns Off.
`trusted_certificates_file`	Returns the file that contains the list of trusted Certificate Authority certificates when the database server acts as a client to an LDAP server.
`tsql_outer_joins`	Returns On if Transact-SQL outer joins can be used in DML statements.
`tsql_variables`	Returns On if you can use the @ sign instead of the colon as a prefix for host variable names in embedded SQL, otherwise, returns Off.
`UncommitOp`	Returns the number of uncommitted operations.
`updatable_statement_isolation`	Returns the isolation level (0, 1, 2, or 3) used by updatable statements when the isolation_level option is set to Readonly-statement-snapshot.
`update_statistics`	Returns On if this connection can send query feedback to the statistics governor. When the update_statistics option is set to Off, the statistics governor does not receive query feedback from the current connection.

Property name	Description
`upgrade_database_capability`	This property is reserved for system use. Do not change the setting of this option.
`user_estimates`	Returns one of the following values that controls whether selectivity estimates in query predicates are respected or ignored by the query optimizer: • **Enabled –** All user-supplied selectivity estimates are respected. You can also use On to turn on this option. • **Override-Magic –** A user selectivity estimate is respected and used only if the optimizer would otherwise choose to use its last-resort, heuristic value (also called the magic value). • **Disabled –** All user estimates are ignored and magic values are used when no other estimate data is available. You can also use Off to turn off this option.
`UserAppInfo`	Returns the string specified by the AppInfo connection parameter in a connection string.
`UserDefinedCounterRate01`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the change in the value of the counter over time.
`UserDefinedCounterRate02`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the change in the value of the counter over time.
`UserDefinedCounterRate03`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the change in the value of the counter over time.

Property name	Description
`UserDefinedCounterRate04`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the change in the value of the counter over time.
`UserDefinedCounterRate05`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the change in the value of the counter over time.
`UserDefinedCounterRaw01`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the absolute value of the counter.
`UserDefinedCounterRaw02`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the absolute value of the counter.
`UserDefinedCounterRaw03`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the absolute value of the counter.
`UserDefinedCounterRaw04`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the absolute value of the counter.

Property name	Description
`UserDefinedCounterRaw05`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the absolute value of the counter.
`UserID`	Returns the user ID for the connection.
`UtilCmdsPermitted`	Returns On or Off to indicate whether utility commands such as CREATE DATABASE, DROP DATABASE, and RESTORE DATABASE are permitted for the connection.
`uuid_has_hyphens`	This property controls the formatting of unique identifier values when they are converted to strings.
`verify_password_function`	Returns the name of the function used for password verification if one has been specified.
`wait_for_commit`	Returns On if the database does not check foreign key integrity until the next COMMIT statement. Otherwise, returns Off and all foreign keys that are not created with the check_on_commit option are checked as they are inserted, updated or deleted.
`WaitStartTime`	Returns the time at which the connection started waiting (or an empty string if the connection is not waiting).
`WaitType`	Returns the reason for the wait, if it is available. Possible values for WaitType are: • **lock –** Returned if the connection is waiting on a lock. • **waitfor –** Returned if the connection is executing a waitfor statement. • **empty string –** Returned if the connection is not waiting, or if the reason for the wait is not available.

Property name	Description
`webservice_namespace_host`	Returns the hostname to be used as the XML namespace within generated WSDL documents if one has been specified.
`webservice_sessionid_name`	Returns the session identifier name that is used by the web server to determine whether session management is being used.

Properties Available for Each Database

You can retrieve the value of a specific database property or the values of all database properties. Database properties apply to an entire database.

The server properties **QueryBypassedCosted**, **QueryBypassedOptimized**, **QueryDescribedOptimizer**, and **StatementPostAnnotatesSimple** are updated only for queries against catalog store tables.

Examples

Use the `db_property` system function to retrieve the value of a database property. The following statement returns the page size of the current database:

```
select db_property ( 'PageSize')
```

Use the `sa_db_properties` system procedure to retrieve the values of all database properties:

```
call sa_db_properties
```

See also

- *PROPERTY Function [System]* on page 301
- *PROPERTY_NAME Function [System]* on page 303
- *PROPERTY_NUMBER Function [System]* on page 303
- *CONNECTION_PROPERTY Function [System]* on page 176

List of database server properties

Database server properties

Property name	Description
`ActiveReq`	Returns the number of server workers that are currently handling client-side requests.

Property name	Description
`ApproximateCPUTime`	Returns an estimate of the amount of CPU time accumulated by the database server, in seconds. The value returned may differ from the actual value by as much as 50%, although typical variations are in the 5-10% range. On multi-processor computers, each CPU (or hyperthread or core) accumulates time, so the sum of accumulated times for all connections may be greater than the elapsed time. This property is supported on Windows and Linux.
`AutoMultiProgrammingLevel`	Returns a value indicating whether the database server is automatically adjusting its multiprogramming level.
`AutoMultiProgrammingLevelStatistics`	Returns a value indicating whether messages about automatic adjustments to the database server's multiprogramming level are displayed in the database server message log.
`AvailIO`	Returns the current number of available I/O control blocks.
`BuildChange`	This property is reserved for system use. Do not change the setting of this property.
`BuildClient`	This property is reserved for system use. Do not change the setting of this property.
`BuildProduction`	Returns Yes if the database server is compiled for production use or returns No if the database server is a debug build.
`BuildReproducible`	This property is reserved for system use. Do not change the setting of this property.
`BytesReceived`	Returns the number of bytes received during client/server communications. This value is updated for HTTP and HTTPS connections.
`BytesReceivedUncomp`	Returns the number of bytes that would have been received during client/server communications if compression was disabled. (This value is the same as the value for BytesReceived if compression is disabled.)

Property name	Description
BytesSent	Returns the number of bytes sent during client/ server communications. This value is updated for HTTP and HTTPS connections.
BytesSentUncomp	Returns the number of bytes that would have been sent during client/server communications if compression was disabled. (This value is the same as the value for BytesSent if compression is disabled.)
CacheAllocated	Returns the number of cache pages that have been allocated for server data structures.
CacheFile	Returns the number of cache pages used to hold data from database files.
CacheFileDirty	Returns the number of cache pages that are dirty (needing a write).
CacheFree	Returns the number of cache pages not being used.
CacheHits	Returns the number of database page lookups.
CachePanics	Returns the number of times the cache manager has failed to find a page to allocate.
CachePinned	Returns the number of pinned cache pages.
CacheRead	Returns the number of cache reads.
CacheReplacements	Returns the number of pages in the cache that have been replaced.
CacheScavenges	Returns the number of times the cache manager has scavenged for a page to allocate.
CacheScavengeVisited	Returns the number of pages visited while scavenging for a page to allocate.
CacheSizingStatistics	Returns Yes if the server is displaying cache sizing statistics when the cache is resized, otherwise, returns No.
CarverHeapPages	Returns the number of heap pages used for short-term purposes, such as query optimization.

Property name	Description
`CharSet`	Returns the CHAR character set in use by the database server.
`ClientStmtCacheHits`	Returns the number of prepares that were not required because of the client statement cache. This is the number of additional prepares that would be required if client statement caching was disabled.
`ClientStmtCacheMisses`	Returns the number of statements in the client statement cache that were prepared again. This is the number of times a cached statement was considered for reuse, but could not be reused because of a schema change, a database option setting, or a DROP VARIABLE statement.
`CollectStatistics`	Returns Yes or No to indicate whether the database server is collecting performance statistics.
`CommandLine`	Returns the command line arguments that were used to start the database server. If the encryption key for a database was specified using the -ek option, the key is replaced with a constant string of asterisks in the value returned by this property.
`Commit`	Returns the number of Commit requests that have been handled.
`CompactPlatformVer`	Returns a condensed version of the PlatformVer property.
`CompanyName`	Returns the name of the company owning this software.

Property name	Description
`ConnectedTime`	Returns the total length of time, in seconds, that all connections have been connected to the database server. The value is updated only when a request completes for a connection or when a connection disconnects. As a result, the value can lag behind for connections that are idle or busy executing for a long time in the database server. The value includes time accrued by any connection, including database events and background server connections (such as the database cleaner).
`ConnsDisabled`	Returns Yes or No to indicate the current setting of the server option to disable new connections.
`ConsoleLogFile`	Returns the name of the file where database server messages are logged if the -o option was specified, otherwise returns an empty string.
`ConsoleLogMaxSize`	Returns the maximum size in bytes of the file used to log database server messages.
`CurrentCacheSize`	Returns the current cache size, in kilobytes.
`CurrentMultiProgrammingLevel`	Returns the current number of tasks that the database server can process concurrently.
`Cursor`	Returns the number of declared cursors that are currently being maintained by the database server.
`CursorOpen`	Returns the number of open cursors that are currently being maintained by the database server.
`DebuggingInformation`	Returns Yes if the server is displaying diagnostic messages for troubleshooting, and No otherwise.
`DefaultCollation`	Returns the collation that would be used for new databases if none is explicitly specified.
`DefaultNcharCollation`	Returns the name of the default NCHAR collation on the server computer (UCA if ICU is installed, and UTF8BIN otherwise).
`DiskRead`	Returns the number of disk reads.

Property name	Description
`DiskReadHintScatterLimit`	Returns the imposed limit on the size (in bytes) of a scatter read hint.
`DiskRetryRead`	Returns the number of disk read retries.
`DiskRetryReadScatter`	Returns the number of disk read retries for scattered reads.
`DiskRetryWrite`	Returns the number of disk write retries.
`DiskSandbox`	Returns On or Off to indicate whether the read-write file operations of the database are restricted to the directory where the main database file is located.
`EventTypeDesc`	Returns the description of the event type associated with a given event type ID.
`EventTypeName`	Returns the system event type name associated with a given event type ID.
`ExchangeTasks`	Returns the number of tasks currently being used for parallel execution of queries.
`ExchangeTasksCompleted`	Returns the total number of internal tasks that have been used for intra-query parallelism since the database server started.
`FipsMode`	Returns Yes if the -fips option was specified when the database server was started, and No otherwise.
`FirstOption`	Returns the number that represents the first connection property that corresponds to a database option.
`FreeBuffers`	Returns the number of available network buffers.
`FunctionMaxParms`	Returns the maximum number of parameters that can be specified by a function. The function is identified by the value specified by the *function-number*, which is a positive integer. For example: `SELECT PROPERTY ( 'FunctionMax-Parms', function-number );` The *function-number* is subject to change between releases.

Property name	Description
`FunctionMinParms`	Returns the minimum number of parameters that must be specified by a function. The function is identified by the value specified by the *function-number*, which is a positive integer. For example: `SELECT PROPERTY ( 'FunctionMinParms', function-number );` The *function-number* is subject to change between releases.
`FunctionName`	Returns the name of the function identified by the value specified by the *function-number* (which is a positive integer): `SELECT PROPERTY ( 'FunctionName', function-number );` The *function-number* is subject to change between releases.
`HeapsCarver`	Returns the number of heaps used for short-term purposes such as query optimization.
`HeapsLocked`	Returns number of relocatable heaps currently locked in the cache.
`HeapsQuery`	Returns the number of heaps used for query processing (hash and sort operations).
`HeapsRelocatable`	Returns the number of relocatable heaps.
`HttpAddresses`	Returns a semicolon delimited list of the TCP/IP addresses the server is listening to for HTTP connections. For example: `(::1):80;127.0.0.1:80`
`HttpNumActiveReq`	Returns the number of HTTP connections that are actively processing an HTTP request. An HTTP connection that has sent its response is not included.
`HttpNumConnections`	Returns the number of HTTP connections that are currently open within the database server. They may be actively processing a request or waiting in a queue of long lived (keep-alive) connections.

Property name	Description
`HttpNumSessions`	Returns the number of active and dormant HTTP sessions within the database server.
`HttpPorts`	Returns the HTTP port numbers for the web server as a comma delimited list.
`HttpsAddresses`	Returns a semicolon delimited list of the TCP/IP addresses the server is listening to for HTTPS connections. For example: `(::1):443;127.0.0.1:443`
`HttpsNumActiveReq`	Returns the number of secure HTTPS connections that are actively processing an HTTPS request. An HTTPS connection that has sent its response is not included.
`HttpsNumConnections`	Returns the number of HTTPS connections that are currently open within the database server. They may be actively processing a request or waiting in a queue of long lived (keep-alive) connections.
`HttpsPorts`	Returns the HTTPS port numbers for the web server as a comma delimited list.
`IdleTimeout`	Returns the default idle timeout.
`IPAddressMonitorPeriod`	Returns the time in seconds that a database server checks for new IP addresses.
`IsAesniAvailable`	Returns Yes if the database server computer's CPU supports the Intel AES-NI instruction set and the computer is running a supported operating system, and No otherwise.
`IsFipsAvailable`	Returns Yes if the FIPS-certified DLL is installed, and No otherwise.
`IsIQ`	This property is reserved for system use. Do not change the setting of this property.
`IsNetworkServer`	Returns Yes if connected to a network database server, and No if connected to a personal database server.

Property name	Description
`IsPortableDevice`	Returns Yes if the database server is running on a laptop, notebook, or other portable device, and No otherwise. Yes is always returned on Windows Mobile. VMWare is not taken into account, so a database server running on a VM that is running on a laptop returns No. On Windows, if it is not possible to determine whether the device is portable, this property returns NULL. This property always returns NULL on Unix.
`IsRsaAvailable`	Returns Yes if the RSA DLL is installed, and No otherwise.
`IsRuntimeServer`	Returns No for all versions of the database server.
`IsService`	Returns Yes if the database server is running as a service, and No otherwise.
`Language`	Returns the locale language for the server.
`LastConnectionProperty`	Returns the number that represents the last connection property.
`LastDatabaseProperty`	Returns the number that represents the last database property.
`LastOption`	Returns the number that represents the last connection property that corresponds to a database option.
`LastServerProperty`	Returns the number that represents the last server property.
`LegalCopyright`	Returns the copyright string for the software.
`LegalTrademarks`	Returns trademark information for the software.
`LicenseCount`	Returns the number of licensed seats or processors.
`LicensedCompany`	Returns the name of the licensed company.
`LicensedUser`	Returns the name of the licensed user.
`LicenseKey`	For internal use only.

Property name	Description
`LicenseType`	Returns the license type. Can be networked seat (per-seat) or CPU-based.
`LivenessTimeout`	Returns the client liveness timeout default.
`LockedCursorPages`	Returns the number of pages used to keep cursor heaps pinned in memory.
`LockedHeapPages`	Returns the number of heap pages locked in the cache.
`MachineName`	Returns the name of the computer running a database server. Typically, this is the computer's host name.
`MainHeapBytes`	Returns the number of bytes used for global server data structures.
`MainHeapPages`	Returns the number of pages used for global server data structures.
`MapPhysicalMemoryEng`	Returns the number of database page address space windows mapped to physical memory in the cache using Address Windowing Extensions.
`MaxCacheSize`	Returns the maximum cache size allowed, in kilobytes.
`MaxConnections`	Returns the maximum number of concurrent connections the server allows. For the personal server, this value defaults to 10. For the network server, this value defaults to about 32000. This value can be lowered using the -gm server option. Computer resources typically limit the number of connections to a network server to a lower value than the default.
`MaxEventType`	Returns the maximum valid event type ID.
`MaxMessage`	This property is deprecated. Returns the current maximum line number that can be retrieved from the database server messages window. This represents the most recent message displayed in the database server messages window.

Property name	Description
`MaxMultiProgrammingLevel`	Returns the maximum number of tasks that the database server can process concurrently.
`MaxRemoteCapability`	Returns the maximum valid capability ID.
`Message`	This property is deprecated. Returns a line from the database server messages window, prefixed by the date and time the message appeared. The second parameter specifies the line number. The value returned by `PROPERTY( "message" )` is the first line of output that was written to the database server messages window. Calling `PROPERTY( "message", n )` returns the n-th line of server output (with zero being the first line). The buffer is finite, so as messages are generated, the first lines are dropped and may no longer be available in memory. In this case, NULL is returned.
`MessageCategoryLimit`	Returns the minimum number of messages of each severity and category that can be retrieved using the sa_server_messages system procedure. The default value is 400.
`MessageText`	This property is deprecated. Returns the text associated with the specified line number in the database server messages window, without a date and time prefix. The second parameter specifies the line number.
`MessageTime`	This property is deprecated. Returns the date and time associated with the specified line number in the database server messages window. The second parameter specifies the line number.
`MessageWindowSize`	This property is deprecated. Returns the maximum number of lines that can be retrieved from the database server messages window.
`MinCacheSize`	Returns the minimum cache size allowed, in kilobytes.
`MinMultiProgrammingLevel`	Returns the minimum number of tasks that the server can process concurrently.

Property name	Description
`MultiPacketsReceived`	Returns the number of multi-packet requests received during client/server communications.
`MultiPacketsSent`	Returns the number of multi-packet responses sent during client/server communications.
`MultiPageAllocs`	Returns the number of multi-page cache allocations.
`MultiProgrammingLevel`	Returns the maximum number of concurrent tasks the server can process. Requests are queued if there are more concurrent tasks than this value. This can be changed with the -gn server option.
`Name`	Returns the alternate name of the server used to connect to the database if one was specified, otherwise, returns the real server name. If the client is connected to a copy node and specified NodeType=COPY in the connection string, then the value of this property may be different than the database server name specified in the client connection string by the ServerName connection parameter.

Property name	Description
`NativeProcessorArchitecture`	Returns a string that identifies the native processor type on which the software is running. For platforms where a processor can be emulated (such as x86 on x64), the actual processor type - not the OS architecture type - is returned. This property does not return a value that indicates whether the operating system is 32-bit or 64-bit. Values can include: Windows Mobile - ARM Solaris - SPARC or X86_64 AIX - PPC HP - IA64 Linux - X86 or X86_64 X86 represents a 32-bit hardware architecture. X86_64 represents a 64-bit hardware architecture. For more information about supported platforms, see *http://www.sybase.com/detail?id=1002288*.
`NumLogicalProcessors`	Returns the number of logical processors (including cores and hyperthreads) enabled on the server computer.
`NumLogicalProcessorsUsed`	Returns the number of logical processors the database server will use. On Windows, use the -gtc option to change the number of logical processors used.
`NumPhysicalProcessors`	Returns the number of physical processors enabled on the server computer. This value is NumLogicalProcessors divided by the number of cores or hyperthreads per physical processor. On some non-Windows platforms, cores or hyperthreads may be counted as physical processors.

Property name	Description
`NumPhysicalProcessorsUsed`	Returns the number of physical processors the database server will use. The personal server is limited to four cores on one processor on some platforms. On Windows, you can use the -gt option to change the number of physical processors used by the network database server.
`ObjectType`	Returns the type of database object. This value is used by the SYSOBJECT system view.
`OmniIdentifier`	This property is reserved for system use. Do not change the setting of this property.
`PacketsReceived`	Returns the number of client/server communication packets received. This value is not updated for HTTP or HTTPS connections.
`PacketsReceivedUncomp`	Returns the number of packets that would have been received during client/server communications if compression was disabled. (This value is the same as the value for PacketsReceived if compression is disabled.)
`PacketsSent`	Returns the number of client/server communication packets sent. This value is not updated for HTTP or HTTPS connections.
`PacketsSentUncomp`	Returns the number of packets that would have been sent during client/server communications if compression was disabled. (This value is the same as the value for PacketsSent if compression is disabled.)
`PageSize`	Returns the size of the database server cache pages. This can be set using the -gp option, otherwise, it is the maximum database page size of the databases specified on the command line.
`PeakCacheSize`	Returns the largest value the cache has reached in the current session, in kilobytes.
`Platform`	Returns the operating system on which the software is running. For example, if you are running on Windows XP, this property returns WindowsXP.

Property name	Description
`PlatformVer`	Returns the operating system on which the software is running, including build numbers, service packs, and so on.
`PrepStmt`	Returns the number of prepared statements currently being maintained by the database server.
`ProcessCPU`	Returns CPU usage for the database server process. Values are in seconds. This property is supported on Windows and Unix. This property is not supported on Windows Mobile. The value returned for this property is cumulative since the database server was started. The value will not match the instantaneous value returned by applications such as the Windows Task Manager or the Windows Performance Monitor.
`ProcessCPUSystem`	Returns system CPU usage for the database server process CPU. This is the amount of CPU time that the database server spent inside the operating system kernel. Values are in seconds. This property is supported on Windows and Unix. This property is not supported on Windows Mobile. The value returned for this property is cumulative since the database server was started. The value will not match the instantaneous value returned by applications such as the Windows Task Manager or the Performance Monitor.
`ProcessCPUUser`	Returns user CPU usage for the database server process. Values are in seconds. This excludes the amount of CPU time that the database server spent inside the operating system kernel. This property is supported on Windows and Unix. This property is not supported on Windows Mobile. The value returned for this property is cumulative since the database server was started. The value will not match the instantaneous value returned by applications such as the Windows Task Manager or the Performance Monitor.
`ProcessID`	Returns the process ID of the database server process.

Property name	Description
`ProcessorAffinity`	Returns the logical processors being used by the database server as specified by the -gta option or by the sa_server_option system procedure and the ProcessorAffinity option.
`ProcessorArchitecture`	Returns a string that identifies the processor type that the current software was built for. Values include: Windows Mobile - ARM Solaris - SPARC or X86_64 AIX - PPC HP - IA64 Linux - X86 or X86_64 X86 represents a 32-bit database server. X86_64 represents a 64-bit database server.
`ProductName`	Returns the name of the software.
`ProductVersion`	Returns the version of the software being run.
`ProfileFilterConn`	Returns the ID of the connection being monitored if procedure profiling for a specific connection is turned on. Otherwise, returns an empty string. You control procedure profiling by user with the sa_server_option procedure.
`ProfileFilterUser`	Returns the user ID being monitored if procedure profiling for a specific user is turned on. Otherwise, returns an empty string. You control procedure profiling by user with the sa_server_option procedure.
`QueryHeapPages`	Returns the number of cache pages used for query processing (hash and sort operations).
`QueryMemActiveCurr`	Returns the number of requests actively using query memory.
`QueryMemActiveEst`	Returns the database server's estimate of the steady state average of the number of requests actively using query memory.

Property name	Description
`QueryMemActiveMax`	Returns the maximum number of requests that are actively allowed to use query memory.
`QueryMemExtraAvail`	Returns the amount of memory available to grant beyond the base memory-intensive grant.
`QueryMemGrantBase`	Returns the minimum amount of memory granted to all requests.
`QueryMemGrantBaseMI`	Returns the minimum amount of memory granted to memory-intensive requests.
`QueryMemGrantExtra`	Returns the number of query memory pages that can be distributed among active memory-intensive requests beyond QueryMemGrantBaseMI.
`QueryMemGrantFailed`	Returns the total number of times a request waited for query memory, but failed to get it.
`QueryMemGrantGranted`	Returns the number of pages currently granted to requests.
`QueryMemGrantRequested`	Returns the total number of times any request attempted to acquire query memory.
`QueryMemGrantWaited`	Returns the total number of times any request waited for query memory.
`QueryMemGrantWaiting`	Returns the current number of requests waiting for query memory.
`QueryMemPages`	Returns the amount of memory that is available for query execution algorithms, expressed as a number of pages.
`QueryMemPercentOfCache`	Returns the amount of memory that is available for query execution algorithms, expressed as a percent of maximum cache size.
`QuittingTime`	Returns the shutdown time for the server. If none is specified, the value is none.
`RememberLastPlan`	Returns Yes if the server is recording the last query optimization plan returned by the optimizer.

Property name	Description
`RememberLastStatement`	Returns Yes if the server is recording the last statement prepared by each connection, and No otherwise.
`RemoteCapability`	Returns the remote capability name associated with a given capability ID.
`RemoteputWait`	Returns the number of times the server had to block while sending a communication packet. Typically, blocking only occurs if the database server is sending data faster than the client or network can receive it. It does not indicate an error condition.
`Req`	Returns the number of times the server has been entered to allow it to handle a new request or continue processing an existing request.
`ReqCountActive`	Returns the number of times the server has been entered to allow it to handle a new request or continue processing an existing request.
`ReqCountBlockContention`	Returns the number of times that any connection has blocked due to contention for an internal server resource.
`ReqCountBlockIO`	Returns the number of times that any connection has blocked while waiting for an IO request to complete.
`ReqCountBlockLock`	Returns the number of times that any connection has blocked while waiting for a row lock held by another connection.
`ReqCountUnscheduled`	Returns the number of times that any connection has blocked while waiting for a server thread to process it.
`ReqTimeActive`	Returns the total amount of time that the server has spent directly servicing requests.
`ReqTimeBlockContention`	Returns the total amount of time that any connection has blocked due to contention for an internal server resource.

Property name	Description
`ReqTimeBlockIO`	Returns the total amount of time that any connection has blocked while waiting for an IO request to complete.
`ReqTimeBlockLock`	Returns the total amount of time that any connection has blocked while waiting for a row lock held by another connection.
`ReqTimeUnscheduled`	Returns the total amount of time that any connection has blocked while waiting for a server thread to process it.
`RequestFilterConn`	Returns the ID of the connection that logging information is being filtered for, otherwise, returns -1.
`RequestFilterDB`	Returns the ID of the database that logging information is being filtered for, otherwise, returns -1.
`RequestLogFile`	Returns the name of the request logging file. An empty string is returned if there is no request logging.
`RequestLogging`	Returns one of SQL, PLAN, HOSTVARS, PROCEDURES, TRIGGERS, OTHER, BLOCKS, REPLACE, ALL, or NONE, indicating the current setting for request logging.
`RequestLogMaxSize`	Returns the maximum size of the request log file.
`RequestLogNumFiles`	Returns the number of request log files being kept.
`RequestsReceived`	Returns the number of client/server communication requests or round trips. It is different from PacketsReceived in that multi-packet requests count as one request, and liveness packets are not included.
`RequestTiming`	Returns Yes if request timing is turned on, and No otherwise. Request timing is turned on using the -zt database server option.
`Rlbk`	The number of rollback requests that have been handled.

Property name	Description
`SendFail`	Returns the number of times that the underlying communications protocols have failed to send a packet.
`ServerEdition`	Returns a space-separated list of words describing the database server type. Values include: Evaluation Developer Web Educational Standard Advanced Workgroup OEM Authenticated If you have a separate license for any of the following features, then the appropriate string(s) are added to the license string that is returned: HighAvailability InMemory FIPS
`ServerName`	Returns the real server name (never an alternate server name). You can use this value to determine which of the operational servers is currently acting as primary in a database mirroring configuration.
`StartDBPermission`	Returns the setting of the -gd server option, which can be one of DBA, all, or none.
`StartTime`	Returns the date/time that the server started.
`StreamsUsed`	Returns the number of database server streams in use.
`TcpIpAddresses`	Returns a semicolon delimited list of the TCP/IP addresses the server is listening to for Command Sequence and TDS connections. For example: `(::1):2638;127.0.0.1:2638`

Property name	Description
`TempDir`	Returns the directory in which temporary files are stored by the server.
`ThreadDeadlocksAvoided`	Returns the number of times a thread deadlock error was detected but not reported to client applications. When the database server starts, the value of this property is 0. To avoid returning thread deadlock errors, the database server dynamically increases the multiprogramming level. If the multiprogramming level cannot be increased, a thread deadlock error is returned to the client application and the ThreadDeadlocksReported property is incremented. The ThreadDeadlocksAvoided property is always 0 on the personal server because dynamic tuning of the multiprogramming level is disabled by default.
`ThreadDeadlocksReported`	Returns the number of times a thread deadlock error was reported to client applications. When the database server starts, the value of this property is 0.
`TimeZoneAdjustment`	Returns the number of minutes that must be added to the Coordinated Universal Time (UTC) to display time local to the server.
`TotalBuffers`	Returns the total number of network buffers.
`UniqueClientAddresses`	Returns the number of unique client network addresses connected to a network server.
`UnschReq`	Returns the number of requests that are currently queued up waiting for an available server worker.
`UserDefinedCounterRate01`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the change in the value of the counter over time.

Property name	Description
`UserDefinedCounterRate02`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the change in the value of the counter over time.
`UserDefinedCounterRate03`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the change in the value of the counter over time.
`UserDefinedCounterRate04`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the change in the value of the counter over time.
`UserDefinedCounterRate05`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the change in the value of the counter over time.
`UserDefinedCounterRaw01`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the absolute value of the counter.
`UserDefinedCounterRaw02`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the absolute value of the counter.

Property name	Description
`UserDefinedCounterRaw03`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the absolute value of the counter.
`UserDefinedCounterRaw04`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the absolute value of the counter.
`UserDefinedCounterRaw05`	Returns the current value of the user-defined performance counter. The semantics of this property are defined by the client application. This counter can also be accessed from the Performance Monitor. The Performance Monitor displays the absolute value of the counter.
`WebClientLogFile`	Returns the name of the web service client log file.
`WebClientLogging`	Returns a value that indicates whether web service client information is being logged to a file.

SQL and External Environment User-Defined Functions

There are two mechanisms for creating user-defined functions in SAP Sybase IQ. You can use the SQL language to write the function, or you can use the CLR, ESQL and ODBC, Java, Perl, or PHP external environments.

Do not confuse SQL UDFs with external C and C++ UDFs. External UDFs require a special license. For information on external UDFs, see the User-Defined Functions manual.

User-Defined Functions in SQL

You can implement your own functions in SQL using the **CREATE FUNCTION** statement. The **RETURN** statement inside the **CREATE FUNCTION** statement determines the data type of the function.

Once you have created a SQL user-defined function, you can use it anywhere a built-in function of the same data type is used.

Note: Avoid using the `CONTAINS` predicate in a view that has a user-defined function, because the `CONTAINS` criteria is ignored. Use the `LIKE` predicate instead, or issue the query outside of a view.

User-Defined Functions in Java

Although SQL functions are useful, Java classes provide a more powerful and flexible way of implementing user-defined functions, with the additional advantage that you can move them from the database server to a client application if desired.

Any *class method* of an installed Java class can be used as a user-defined function anywhere a built-in function of the same data type is used.

Instance methods are tied to particular instances of a class, and so have different behavior from standard user-defined functions.

For more information on creating Java classes, and on class methods, see *Java in the Database* in the *Programming* guide.

Miscellaneous Functions

Miscellaneous functions perform operations on arithmetic, string, or date/time expressions, including the return values of other functions.

This table lists the miscellaneous functions and their parameters.

Miscellaneous Functions	Parameters
ARGN	(*integer-expr*, *expression* [, …)
COALESCE	(*expression*, *expression* [, *expression* …])
IFNULL	(*expression1*, *expression2* [, *expression3*])
ISNULL	(*expression*, *expression* [, *expression* …])
NULLIF	(*expression1*, *expression2*)
NUMBER	(*)
ROWID	(*table-name*)

Compatibility
Adaptive Server Enterprise supports only the COALESCE, ISNULL, and NULLIF functions.

Alphabetical List of Functions

This section describes each SQL function individually.

The function type, for example, Numeric or String, is indicated in brackets next to the function name.

Some of the results in the examples have been rounded or truncated.

The actual values of database object IDs, such as the object ID of a table or the column ID of a column, might differ from the values shown in the examples.

ABS Function [Numeric]

Returns the absolute value of a numeric expression.

Syntax

```
ABS ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	The number whose absolute value is to be returned.

Returns

An absolute value of the numeric expression.

Numeric-expression Data Type	Returns
INT	INT
FLOAT	FLOAT
DOUBLE	DOUBLE
NUMERIC	NUMERIC

Example

The following statement returns the value 66:

```
SELECT ABS( -66 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

ACOS Function [Numeric]

Returns the arc-cosine, in radians, of a numeric expression.

Syntax

```
ACOS ( numeric-expression )
```

Parameters

Table 14. Parameters

Parameter	Description
numeric-expression	The cosine of the angle.

Returns

DOUBLE

Example

The following statement returns the value 1.023945:

```
SELECT ACOS( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *COS Function [Numeric]* on page 183
- *ATAN2 Function [Numeric]* on page 162
- *ATAN Function [Numeric]* on page 161
- *ASIN Function [Numeric]* on page 160
- *COT Function [Numeric]* on page 184
- *SIN Function [Numeric]* on page 335
- *TAN Function [Numeric]* on page 363

ARGN Function [Miscellaneous]

Returns a selected argument from a list of arguments.

Syntax

```
ARGN ( integer-expression, expression [ , …] )
```

Parameters

Table 15. Parameters

Parameter	Description
integer-expression	The position of an argument within a list of expressions.
expression	n expression of any data type passed into the function. All supplied expressions must be of the same data type.

Returns

Using the value of the *integer-expression* as *n*, returns the nth argument (starting at 1) from the remaining list of arguments.

Example

The following statement returns the value 6:

```
SELECT ARGN( 6, 1,2,3,4,5,6 ) FROM iq_dummy
```

Usage

Using the value of *integer-expression* as *n* returns the *n*th argument (starting at 1) from the remaining list of arguments. While the expressions can be of any data type, they must all be of the same data type. The integer expression must be from one to the number of expressions in the list or NULL is returned. Multiple expressions are separated by a comma.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

ASCII Function [String]

Returns the integer ASCII value of the first byte in a string-expression.

Syntax

```
ASCII ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string.

Returns
SMALLINT

Example
The following statement returns the value 90, when the collation sequence is set to the default ISO_BINENG:

```
SELECT ASCII( 'Z' ) FROM iq_dummy
```

Usage
If the string is empty, **ASCII** returns zero. Literal strings must be enclosed in quotes.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

ASIN Function [Numeric]

Returns the arc-sine, in radians, of a number.

Syntax

```
ASIN ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	The sine of the angle.

Returns
DOUBLE

Example
The following statement returns the value 0.546850.

```
SELECT ASIN( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also
- *COS Function [Numeric]* on page 183
- *ATAN2 Function [Numeric]* on page 162

- *ATAN Function [Numeric]* on page 161
- *ACOS Function [Numeric]* on page 158
- *COT Function [Numeric]* on page 184
- *SIN Function [Numeric]* on page 335
- *TAN Function [Numeric]* on page 363

ATAN Function [Numeric]

Returns the arc-tangent, in radians, of a number.

Syntax

```
ATAN ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	The tangent of the angle.

Returns

DOUBLE

Example

The following statement returns the value 0.479519:

```
SELECT ATAN( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *COS Function [Numeric]* on page 183
- *ATAN2 Function [Numeric]* on page 162
- *ASIN Function [Numeric]* on page 160
- *ACOS Function [Numeric]* on page 158
- *COT Function [Numeric]* on page 184
- *SIN Function [Numeric]* on page 335
- *TAN Function [Numeric]* on page 363

ATAN2 Function [Numeric]

Returns the arc-tangent, in radians, of the ratio of two numbers.

Syntax

```
ATAN2 ( numeric-expression1, numeric-expression2 )
```

Parameters

Parameter	Description
numeric-expression1	The numerator in the ratio whose arc tangent is calculated.
numeric-expression2	The denominator in the ratio whose arc-tangent is calculated.

Returns

DOUBLE

Example

The following statement returns the value 0.00866644968879073143:

```
SELECT ATAN2( 0.52, 060 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—ATAN2 is not supported by Adaptive Server Enterprise.

See also

- *COS Function [Numeric]* on page 183
- *ATAN Function [Numeric]* on page 161
- *ASIN Function [Numeric]* on page 160
- *ACOS Function [Numeric]* on page 158
- *COT Function [Numeric]* on page 184
- *SIN Function [Numeric]* on page 335
- *TAN Function [Numeric]* on page 363

AVG Function [Aggregate]

Computes the average of a numeric expression for a set of rows, or computes the average of a set of unique values.

Syntax

```
AVG ( numeric-expression  | DISTINCT column-name )
```

Parameters

Parameter	Description
numeric-expression	The value whose average is calculated over a set of rows.
DISTINCT column-name	Computes the average of the unique values in column-name. This is of limited usefulness, but is included for completeness

Returns

Returns the NULL value for a group containing no rows.

Returns DOUBLE if the argument is DOUBLE, otherwise NUMERIC.

Example

The following statement returns the value 49988.6:

```
SELECT AVG ( salary ) FROM Employees
```

Usage

This average does not include rows where *numeric -expression* is the NULL value. Returns the NULL value for a group containing no rows.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *COUNT Function [Aggregate]* on page 187
- *SUM Function [Aggregate]* on page 360

BFILE Function [Data extraction]

Extracts individual `LONG BINARY` and `LONG VARCHAR` cells to individual operating system files on the server.

Usage

The IQ data extraction facility includes the **BFILE** function, which allows you to extract individual `LONG BINARY` and `LONG VARCHAR` cells to individual operating system files on the server. You can use **BFILE** with or without the data extraction facility.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

BIGINTTOHEX Function [Data Type Conversion]

Returns the hexadecimal equivalent in `VARCHAR(16)` of a decimal integer.

Syntax

```
BIGINTTOHEX ( integer-expression )
```

Parameters

Parameter	Description
integer-expression	The integer to be converted to hexadecimal.

Examples

Returns the value `0000000000000009`:

```
SELECT BIGINTTOHEX(9) FROM iq_dummy
```

Returns the value `FFFFFFFFFFFFFFF7`:

```
SELECT BIGINTTOHEX (-9) FROM iq_dummy
```

Usage

BIGINTTOHEX accepts an integer expression that evaluates to `BIGINT` and returns the hexadecimal equivalent. Returned values are left appended with zeros up to a maximum of 16 digits. All types of unscaled integer data types are accepted as integer expressions.

Conversion is done automatically, if required. Constants are truncated, only if the fraction values are zero. A column cannot be truncated, if the column is declared with a positive scale value. If conversion fails, SAP Sybase IQ returns an error unless the `CONVERSION_ERROR` option is `OFF`. In that case, the result is `NULL`.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *HEXTOBIGINT Function [Data Type Conversion]* on page 231
- *HEXTOINT Function [Data Type Conversion]* on page 233
- *INTTOHEX Function [Data Type Conversion]* on page 248

BIT_LENGTH Function [String]

Returns an unsigned 64-bit value containing the bit length of the column parameter.

Syntax

```
BIT_LENGTH( column-name )
```

Parameters

Parameter	Description
column-name	The name of a column

Returns

INT

Usage

The return value of a NULL argument is NULL.

The **BIT_LENGTH** function supports all SAP Sybase IQ data types.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *Function Support* in *Unstructured Data Analytics*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by SQL Anywhere or Adaptive Server Enterprise.

See also

- *BYTE_LENGTH Function [String]* on page 166
- *CHAR_LENGTH Function [String]* on page 171
- *COL_LENGTH Function [System]* on page 175
- *DATALENGTH Function [System]* on page 189
- *LEN Function [String]* on page 260
- *LENGTH Function [String]* on page 261
- *OBJECT_NAME Function [System]* on page 291
- *OCTET_LENGTH Function [String]* on page 292
- *STR_REPLACE Function [String]* on page 352

BYTE_LENGTH Function [String]

Returns the number of bytes in a string.

Syntax

```
BYTE_LENGTH ( string-expression )
```

Parameters

Parameters	Description
string-expression	The string whose length is to be calculated.

Returns

INT

Example

Returns the value 12:

```
SELECT BYTE_LENGTH( 'Test Message' ) FROM iq_dummy
```

Usage

Trailing white space characters are included in the length returned.

The return value of a NULL string is NULL.

If the string is in a multibyte character set, the **BYTE_LENGTH** value differs from the number of characters returned by **CHAR_LENGTH**.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data:

- The **BYTE_LENGTH** function supports both `LONG BINARY` columns and variables and `LONG VARCHAR` columns and variables, only if the query returns less than 2GB. If the byte length of the returned `LONG BINARY` or `LONG VARCHAR` data is greater than or equal to 2GB, **BYTE_LENGTH** returns an error that says you must use the **BYTE_LENGTH64** function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- SAP Sybase—Not supported by Adaptive Server Enterprise.

See also

- *BIT_LENGTH Function [String]* on page 165
- *CHAR_LENGTH Function [String]* on page 171
- *COL_LENGTH Function [System]* on page 175

- *DATALENGTH Function [System]* on page 189
- *LEN Function [String]* on page 260
- *LENGTH Function [String]* on page 261
- *OBJECT_NAME Function [System]* on page 291
- *OCTET_LENGTH Function [String]* on page 292
- *STR_REPLACE Function [String]* on page 352

BYTE_LENGTH64 Function

BYTE_LENGTH64 returns an unsigned 64-bit value containing the byte length of the `LONG BINARY` column parameter.

Usage

BYTE_LENGTH64 also supports the `LONG VARCHAR` data type and `LONG BINARY` and `LONG VARCHAR` variables of any data size.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *Function Support* in *Unstructured Data Analytics*.

BYTE_SUBSTR64 and BYTE_SUBSTR Functions

BYTE_SUBSTR64 and **BYTE_SUBSTR** return the long binary byte substring of the `LONG BINARY` column parameter.

Usage

The **BYTE_SUBSTR64** and **BYTE_SUBSTR** functions also support the `LONG VARCHAR` data type and `LONG BINARY` and `LONG VARCHAR` variables of any data size.

CHAR_LENGTH64 also supports `LONG VARCHAR` variables of any data size.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *BYTE_SUBSTR64 and BYTE_SUBSTR Functions* in *Unstructured Data Analytics*.

CAST Function [Data Type Conversion]

Returns the value of an expression converted to a supplied data type.

Syntax

```
CAST ( expression AS data type )
```

Parameters

Parameters	Description
expression	The expression to be converted.
data type	The target data type.

Returns

The specified data type.

Examples

The following function ensures a string is used as a date:

```
CAST( '2000-10-31' AS DATE )
```

The value of the expression **1 + 2** is calculated, and the result cast into a single-character string, the length the data server assigns:

```
CAST( 1 + 2 AS CHAR )
```

You can use the **CAST** function to shorten strings:

```
SELECT CAST( lname AS CHAR(5) ) FROM Customers
```

Usage

If you do not indicate a length for character string types, SAP Sybase IQ chooses an appropriate length. If neither precision nor scale is specified for a DECIMAL conversion, the database server selects appropriate values.

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

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

is described as:

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

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *CONVERT Function [Data Type Conversion]* on page 178
- *HOURS Function [Date and Time]* on page 235
- *MINUTES Function [Date and Time]* on page 275

- *MONTHS Function [Date and Time]* on page 278
- *REPLACE Function [String]* on page 321
- *SECOND Function [Date and Time]* on page 331
- *WEEKS Function [Date and Time]* on page 375
- *YEAR Function [Date and Time]* on page 380
- *YEARS Function [Date and Time]* on page 381
- *DAYS Function [Date and Time]* on page 208

CEIL Function [Numeric]

Returns the smallest integer greater than or equal to the specified expression.

CEIL is as synonym for **CEILING**.

Syntax

```
CEIL ( numeric-expression )
```

Parameters

Parameters	Description
expression	A column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, **CEIL** generates an error. The return value has the same data type as the value supplied.

Usage

For a given expression, the **CEIL** function takes one argument. For example, **CEIL (-123.45)** returns -123. **CEIL (123.45)** returns 124.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *FLOOR Function [Numeric]* on page 226
- *CEILING Function [Numeric]* on page 169

CEILING Function [Numeric]

Returns the ceiling (smallest integer not less than) of a number.

CEIL is as synonym for **CEILING**.

Syntax

```
CEILING ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	The number whose ceiling is to be calculated.

Returns

DOUBLE

Examples

The following statement returns the value 60.00000:

```
SELECT CEILING( 59.84567 ) FROM iq_dummy
```

The following statement returns the value 123:

```
SELECT CEILING( 123 ) FROM iq_dummy
```

The following statement returns the value 124.00:

```
SELECT CEILING( 123.45 ) FROM iq_dummy
```

The following statement returns the value -123.00:

```
SELECT CEILING( -123.45 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *FLOOR Function [Numeric]* on page 226
- *CEIL Function [Numeric]* on page 169

CHAR function [String]

Returns the character with the ASCII value of a number.

Syntax

```
CHAR ( integer-expression )
```

Parameters

Parameter	Description
integer-expression	The number to be converted to an ASCII character. The number must be in the range 0 to 255, inclusive.

Returns
VARCHAR

Examples
The following statement returns the value "Y":

```
SELECT CHAR( 89 ) FROM iq_dummy
```

The following statement returns the value "S":

```
SELECT CHAR( 83 ) FROM iq_dummy
```

Usage
The character in the current database character set corresponding to the supplied numeric expression modulo 256 is returned.

CHAR returns NULL for integer expressions with values greater than 255 or less than zero.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

CHAR_LENGTH Function [String]

Returns the number of characters in a string.

Syntax

```
CHAR_LENGTH ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string whose length is to be calculated.

Returns
INT

Usage

Trailing white space characters are included in the length returned.

The return value of a `NULL` string is NULL.

If the string is in a multibyte character set, the **CHAR_LENGTH** value may be less than the **BYTE_LENGTH** value.

CHAR_LENGTH64 also supports `LONG VARCHAR` variables of any data size.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data. See *CHAR_LENGTH Function* in *Unstructured Data Analytics*.

Example

The following statement returns the value 8:

```
SELECT CHAR_LENGTH( 'Chemical' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *BIT_LENGTH Function [String]* on page 165
- *BYTE_LENGTH Function [String]* on page 166
- *COL_LENGTH Function [System]* on page 175
- *DATALENGTH Function [System]* on page 189
- *LEN Function [String]* on page 260
- *LENGTH Function [String]* on page 261
- *OBJECT_NAME Function [System]* on page 291
- *OCTET_LENGTH Function [String]* on page 292
- *STR_REPLACE Function [String]* on page 352

CHAR_LENGTH64 Function

The **CHAR_LENGTH64** function returns an unsigned 64-bit value containing the character length of the `LONG VARCHAR` column parameter, including the trailing blanks.

Usage

CHAR_LENGTH64 also supports `LONG VARCHAR` variables of any data size.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data. See *CHAR_LENGTH64 Function* in *Unstructured Data Analytics*.

CHARINDEX Function [String]

Returns the position of the first occurrence of a specified string in another string.

Syntax

```
CHARINDEX ( string-expression1, string-expression2 )
```

Parameters

Parameter	Description
string-expression1	The string for which you are searching. This string is limited to 255 bytes.
string-expression2	The string to be searched. The position of the first character in the string being searched is 1.

Returns

INT

Example

The statement:

```
SELECT Surname, GivenName
FROM Employees
WHERE CHARINDEX('K', Surname ) = 1
```

returns the following values:

Surname	GivenName
Klobucher	James
Kuo	Felicia
Kelly	Moira

Usage

All the positions or offsets, returned or specified, in the **CHARINDEX** function are always character offsets and may be different from the byte offset for multibyte data.

If the string being searched contains more than one instance of the specified string, **CHARINDEX** returns the position of the first instance.

If the string being searched does not contain the specified string, **CHARINDEX** returns zero (0).

Searching for a zero-length string returns 1.

If any of the arguments is NULL, the result is NULL.

CHARINDEX returns a 32 bit signed integer position for `CHAR` and `VARCHAR` columns.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data. See *CHARINDEX Function* in *Unstructured Data Analytics* .

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *SUBSTRING Function [String]* on page 357

COALESCE Function [Miscellaneous]

Returns the first non-NULL expression from a list.

Syntax

```
COALESCE ( expression, expression [ , … ] )
```

Parameters

Parameter	Description
expression	Any expression.

Returns

ANY

Example

The following statement returns the value 34:

```
SELECT COALESCE( NULL, 34, 13, 0 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *ISNULL Function [Miscellaneous]* on page 250

COL_LENGTH Function [System]

Returns the defined length of a column.

Syntax

```
COL_LENGTH ( table-name, column-name )
```

Parameter	Description
table-name	The table name.
column-name	The column name.

Example

Returns the column length 35:

```
SELECT COL_LENGTH ( 'CUSTOMERS', 'ADDRESS' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ.

See also

- *BIT_LENGTH Function [String]* on page 165
- *BYTE_LENGTH Function [String]* on page 166
- *CHAR_LENGTH Function [String]* on page 171
- *DATALENGTH Function [System]* on page 189
- *LEN Function [String]* on page 260
- *LENGTH Function [String]* on page 261
- *OBJECT_NAME Function [System]* on page 291
- *OCTET_LENGTH Function [String]* on page 292
- *STR_REPLACE Function [String]* on page 352

COL_NAME Function [System]

Returns the column name.

Syntax

```
COL_NAME ( table-id, column-id [ , database-id ] )
```

Parameters

Table 16. Parameters

Parameter	Description
table-id	The object ID of the table.
column-id	The column ID of the column.
database-id	The database ID.

Examples

The following statement returns the column name `lname`. The object ID of the `Customers` table is 100209, as returned by the **OBJECT_ID** function. The column ID is stored in the `column_id` column of the `syscolumn` system table. The database ID of the `iqdemo` database is 0, as returned by the DB_ID function.

```
SELECT COL_NAME( 100209, 3, 0 ) FROM iq_dummy
```

The following statement returns the column name **city**.

```
SELECT COL_NAME ( 100209, 5 )FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ.

See also

- *DB_ID Function [System]* on page 210
- *DB_NAME Function [System]* on page 211
- *DB_PROPERTY Function [System]* on page 212
- *NEXT_DATABASE Function [System]* on page 283
- *OBJECT_ID Function [System]* on page 290
- *OBJECT_NAME Function [System]* on page 291

CONNECTION_PROPERTY Function [System]

Returns the value of a given connection property as a string.

Syntax

```
CONNECTION_PROPERTY ( { integer-expression1 | string-expression }
                                                  … [ , integer-
expression2 ] )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
integer-expression1	In most cases, it is more convenient to supply a string expression as the first argument. If you do supply integer-expression1, it is the connection property ID. You can determine this using the PROPERTY_NUMBER function.
string-expression	The connection property name. You must specify either the property ID or the property name.
integer-expression2	The connection ID of the current database connection. The current connection is used if this argument is omitted.

Returns
VARCHAR

Example
The following statement returns the number of prepared statements being maintained, for example, 4:

```
SELECT connection_property( 'PrepStmt' )FROM iq_dummy
```

Usage
The current connection is used if the second argument is omitted.

Standards and Compatibility

- Vendor extension to ISO/ANSI SQL grammar.
- Compatible with Adaptive Server Enterprise.

See also

- *PROPERTY Function [System]* on page 301
- *PROPERTY_NAME Function [System]* on page 303
- *PROPERTY_NUMBER Function [System]* on page 303
- *Properties Available for the Server* on page 101
- *Properties Available for Each Database* on page 133
- *Connection Properties* on page 100
- *sp_iqshowpsexe Procedure* on page 557
- *sp_iqcontext Procedure* on page 457

CONVERT Function [Data Type Conversion]

Returns an expression converted to a supplied data type.

Syntax

```
CONVERT ( data-type, expression [ , format-style ] )
```

Parameters

Parameter	Description
data-type	The data type to which the expression is converted.
expression	The expression to be converted.
format-style	For converting strings to date or time data types and vice versa, format-style is a style code number that describes the date format string to be used.

If no *format-style* argument is provided, the database option settings are used.

Table 17. CONVERT format style code output

Without century (yy)	With century (yyyy)	Output
-	0 or 100	mmm dd yyyy hh:nnAM (or PM)
1	101	mm/dd/yy[yy]
2	102	[yy]yy.mm.dd
3	103	dd/mm/yy[yy]
4	104	dd.mm.yy[yy]
5	105	dd-mm-yy[yy]
6	106	dd mmm yy[yy]
7	107	mmm dd, yy[yy]
8	108	hh:nn:ss
-	9 or 109	mmm dd yyyy hh:nn:ss:sssAM (or PM)
10	110	mm-dd-yy[yy]

Without century (yy)	With century (yyyy)	Output
11	111	[yy]yy/mm/dd
12	112	[yy]yymmdd
-	13 or 113	dd mmm yyyy hh:nn:ss:sss (24 hour clock, Europe default + milliseconds, 4-digit year)
14	114	hh:nn:ss (24 hour clock)
-	20 or 120	yyyy-mm-dd hh:nn:ss (24-hour clock, ODBC canonical, 4-digit year)
-	21 or 121	yyyy-mm-dd hh:nn:ss.sss (24 hour clock, ODBC canonical with milliseconds, 4-digit year)
36	136	hh:nn:ss.ssssssAM (or PM)
37	137	hh:nn:ss.ssssss
38	138	mmm dd yy[yy] hh:nn:ss.ssssssAM (or PM)
39	139	mmm dd yy[yy] hh:nn:ss.ssssss
40	140	[yy]yy-mm-dd hh:nn:ss.ssssss
-	365	yyyyjjj (as a string or integer, where jjj is the Julian day number from 1 to 366 within the year)

Abbreviations and values for date parts in the **CONVERT** format style table:

Abbreviation	Date part	Values
hh	hour	0 – 23
nn	minute	0 – 59
ss	second	0 – 59
sss	millisecond	0 – 999
ssssss	microsecond	0 – 999999
mmm	month	Jan to Dec
dd	day	1 – 31
yyyy	year	0001 – 9999
mm	month	1 – 12

Returns

The data type specified.

Examples

The following statements illustrate the use of format styles:

```
SELECT CONVERT( CHAR( 20 ), order_date, 104 )
FROM sales_order
```

order_date
16.03.1993
20.03.1993
23.03.1993
25.03.1993
...

```
SELECT CONVERT( CHAR( 20 ), order_date, 7 )
FROM sales_order
```

order_date
mar 16, 93
mar 20, 93
mar 23, 93
mar 25, 93
...

```
SELECT order_datetime, CONVERT(CHAR(30), order_datetime, 40)
order_datetime40,
CONVERT(CHAR(30), order_datetime, 140) order_datetime140
FROM sales_order;
```

order_datetime	order_datetime40	order_datetime140
03/05/2009 01:03.05.123456	09-03-05 01:03:05.123456	2009-03-05 01:03:05.123456
03/05/2009 13:05.07.654321	09-03-05 13:05:07.654321	2009-03-05 13:05:07.654321

`SELECT CONVERT(CHAR(50), DATETIME('2009-11-03 11:10:42.033189'), 136) FROM iq_dummy` returns `11:10:42.033189AM`

`SELECT CONVERT(CHAR(50), NOW(), 137) FROM iq_dummy` returns `14:54:48.794122`

The following statements illustrate the use of the format style 365, which converts data of type `DATE` and `DATETIME` to and from either string or integer type data:

```
CREATE TABLE tab
   (date_col DATE, int_col INT, char7_col CHAR(7));
INSERT INTO tab (date_col, int_col, char7_col)
   VALUES ('Dec 17, 2004', 2004352, '2004352');
```

`SELECT CONVERT(VARCHAR(8), tab.date_col, 365) FROM tab;` returns `'2004352'`

`SELECT CONVERT(INT, tab.date_col, 365) from tab;` returns `2004352`

`SELECT CONVERT(DATE, tab.int_col, 365) FROM TAB;` returns `2004-12-17`

`SELECT CONVERT(DATE, tab.char7_col, 365) FROM tab;` returns `2004-12-17`

The following statement illustrates conversion to an integer, and returns the value 5.

```
SELECT CONVERT( integer, 5.2 ) FROM iq_dummy
```

Usage

The result data type of a **CONVERT** function is a `LONG VARCHAR`. If you use **CONVERT** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **CONVERT** to the correct data type and size.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise and SQL Anywhere, except for format style 365, which is an SAP Sybase IQ -only extension.

See also

- *CAST Function [Data Type Conversion]* on page 167
- *HOURS Function [Date and Time]* on page 235
- *MINUTES Function [Date and Time]* on page 275
- *MONTHS Function [Date and Time]* on page 278
- *REPLACE Function [String]* on page 321
- *SECOND Function [Date and Time]* on page 331
- *WEEKS Function [Date and Time]* on page 375
- *YEAR Function [Date and Time]* on page 380
- *YEARS Function [Date and Time]* on page 381

CORR Function [Aggregate]

Returns the correlation coefficient of a set of number pairs.

Syntax 1

```
CORR (dependent-expression, independent-expression)
```

Syntax 2

```
CORR (dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Table 18. Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent-expression.
independent-expression	The variable that influences the outcome.

Returns
DOUBLE

Example
The following example performs a correlation to discover whether age is associated with income level. This function returns the value 0.440227:

```
SELECT CORR( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) ) FROM
Employees;
```

Usage
The **CORR** function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then CORR returns NULL.

dependent-expression and independent-expression are both numeric. The function is applied to the set of (dependent-expression, independent-expression) after eliminating the pairs for which either dependent-expression or independent-expression is NULL. The following computation is made:

COVAR_POP (y, x) / (STDDEV_POP (x) * STDDEV_POP (y))

where x represents the dependent-expression and y represents the independent-expression.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the **SELECT** statement.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL foundation feature outside of core SQL.
- Sybase—Compatible with SQL Anywhere.

See also

- *Windowing Aggregate Function Usage* on page 83

COS Function [Numeric]

Returns the cosine of a number, expressed in radians.

Syntax

```
COS ( numeric-expression )
```

Parameters

Table 19. Parameters

Parameter	Description
numeric-expression	The angle, in radians.

Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Example

The following statement returns the value 0.86781:

```
SELECT COS( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *ATAN2 Function [Numeric]* on page 162
- *ATAN Function [Numeric]* on page 161
- *ASIN Function [Numeric]* on page 160

- *ACOS Function [Numeric]* on page 158
- *COT Function [Numeric]* on page 184
- *SIN Function [Numeric]* on page 335
- *TAN Function [Numeric]* on page 363

COT Function [Numeric]

Returns the cotangent of a number, expressed in radians.

Syntax

```
COT ( numeric-expression )
```

Parameters

Table 20. Parameters

Parameter	Description
numeric-expression	The angle, in radians.

Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Example

The following statement returns the value 1.74653:

```
SELECT COT( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *COS Function [Numeric]* on page 183
- *ATAN2 Function [Numeric]* on page 162
- *ATAN Function [Numeric]* on page 161
- *ASIN Function [Numeric]* on page 160
- *ACOS Function [Numeric]* on page 158
- *SIN Function [Numeric]* on page 335
- *TAN Function [Numeric]* on page 363

COVAR_POP Function [Aggregate]

Returns the population covariance of a set of number pairs.

Syntax 1

```
COVAR_POP (dependent-expression, independent-expression)
```

Syntax 2

```
COVAR_POP (dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Example

The following example measures the strength of association between employee age and salary. This function returns the value 73785.840059:

```
SELECT COVAR_POP( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) )
FROM Employees;
```

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **COVAR_POP** returns NULL.

dependent-expression and independent-expression are both numeric. The function is applied to the set of (dependent-expression, independent-expression) after eliminating the pairs for which either dependent-expression or independent-expression is NULL. The following computation is made:

(SUM(x*y) - SUM(x) * SUM(y) / n) / n

where x represents the dependent-expression and y represents the independent-expression.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL foundation feature outside of core SQL.
- Sybase—Compatible with SQL Anywhere.

COVAR_SAMP Function [Aggregate]

Returns the sample covariance of a set of number pairs.

Syntax 1

COVAR_SAMP (*dependent-expression*, *independent-expression*)

Syntax 2

COVAR_SAMP (*dependent-expression*, *independent-expression*)

OVER (*window-spec*)

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Example

The following example measures the strength of association between employee age and salary. This function returns the value 74782.946005:

```
SELECT COVAR_SAMP( Salary, ( 2008 - YEAR( BirthDate ) ) ) FROM
Employees;
```

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **COVAR_SAMP** returns NULL.

Both dependent-expression and independent-expression are numeric. The function is applied to the set of (dependent-expression, independent-expression) after eliminating the pairs for which either dependent-expression or independent-expression is NULL.

```
(SUM(x*y) - SUM(x) * SUM(y) / n) / (n-1)
```

where x represents the dependent-expression and y represents the independent-expression.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL foundation feature outside of core SQL.
- Sybase—Compatible with SQL Anywhere.

COUNT Function [Aggregate]

Counts the number of rows in a group, depending on the specified parameters.

Syntax

```
COUNT ( * | expression | DISTINCT column-name )
```

Parameters

Parameter	Description
*	Returns the number of rows in each group.
expression	Returns the number of rows in each group where expression is not the NULL value.
DISTINCT column-name	Returns the number of different values in column-name. Rows where the value is the NULL value are not included in the count.

Note: When the query results are displayed, the * is not displayed in the column header, and appears as:

```
Count()
```

Returns

INT

Example

Returns each unique city, and the number of rows with that city value:

```
SELECT city , Count(*)
FROM Employees
GROUP BY city
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *AVG Function [Aggregate]* on page 162
- *SUM Function [Aggregate]* on page 360
- *Windowing Aggregate Function Usage* on page 83

CUME_DIST Function [Analytical]

The **CUME_DIST** function is a rank analytical function that calculates the relative position of one value among a group of rows. It returns a decimal value between 0 and 1.

Syntax

```
CUME_DIST () OVER (window-spec)
```

Returns

A DOUBLE value between 0 and 1

Example

The following example returns a result set that provides a cumulative distribution of the salaries of employees who live in California:

```
SELECT DepartmentID, Surname, Salary,CUME_DIST() OVER (PARTITION BY
DepartmentIDORDER BY Salary DESC) "Rank"FROM Employees WHERE State IN
('CA');
```

The returned result set is:

Table 21. CUME_DIST result set

DepartmentID	Surname	Salary	Rank
200	Savarino	72,300.000	0.333333
200	Clark	45,000.000	0.666667
200	Overbey	39,300.000	1.000000

Usage

SAP Sybase IQ calculates the cumulative distribution of a value of x in a set S of size N using:CUME_DIST(x) = number of values in S coming before and including x in the specified order / N

Composite sort-keys are not currently allowed in the **CUME_DIST** function. You can use composite sort-keys with any of the other rank functions.

You can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement. The *window-spec* must contain the **ORDER BY** clause, and cannot contain a **ROWS** or **RANGE** clause.

Note: DISTINCT is not supported.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL feature T612.
- Sybase—Compatible with SQL Anywhere.

DATALENGTH Function [System]

Returns the length of the expression in bytes.

Syntax

```
DATALENGTH ( expression )
```

Parameters

Parameter	Description
expression	The expression is usually a column name. If the expression is a string constant, it must be enclosed in quotes.

Returns

UNSIGNED INT

Usage

Table 22. DATALENGTH return values

Data type	DATALENGTH
SMALLINT	2
INTEGER	4
DOUBLE	8
CHAR	Length of the data
BINARY	Length of the data

Example

Returns the value 35, the longest string in the `company_name` column:

```
SELECT MAX( DATALENGTH( company_name ) )
FROM Customers
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ.

See also

- *BIT_LENGTH Function [String]* on page 165
- *BYTE_LENGTH Function [String]* on page 166
- *CHAR_LENGTH Function [String]* on page 171
- *COL_LENGTH Function [System]* on page 175
- *LEN Function [String]* on page 260
- *LENGTH Function [String]* on page 261
- *OBJECT_NAME Function [System]* on page 291
- *OCTET_LENGTH Function [String]* on page 292
- *STR_REPLACE Function [String]* on page 352

DATE Function [Date and Time]

Converts the expression into a date, and removes any hours, minutes, or seconds.

Syntax

```
DATE ( expression )
```

Parameters

Table 23. Parameters

Parameter	Description
expression	The value to be converted to date format. The expression is usually a string.

Returns

DATE

Example

The following statement returns the value 1988-11-26 as a date.

```
SELECT DATE( '1988-11-26 21:20:53' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

DATEADD Function [Date and Time]

Returns the date produced by adding the specified number of the specified date parts to a date.

Syntax

```
DATEADD ( date-part, numeric-expression, date-expression )
```

Parameters

Parameter	Description
date part	The date part to be added to the date.
numeric-expression	The number of date parts to be added to the date. The numeric-expression can be any numeric type; the value is truncated to an integer. The maximum microsecond in numeric-expression is 2147483647, that is, 35:47.483647 (35 mins 47 secs 483647 mcs).
date-expression	The date to be modified.

Returns

TIMESTAMP

Example

The following statement returns the value 1995-11-02 00:00:00.000:

```
SELECT DATEADD( MONTH, 102, '1987/05/02' ) FROM iq_dummy
```

The following statement returns the value 2009-11-10 14:57:52.722016:

```
SELECT DATEADD(MICROSECOND, 15, '2009-11-10
14:57:52.722001') FROM iq_dummy
```

The following statement returns the value 1985-05-02 00:00:00.123456:

```
SELECT DATEADD(MICROSECOND, 123456, '1985/05/02')
FROM iq_dummy
```

The following statement returns the value 1985-05-01 23:59:59.876544:

```
SELECT DATEADD(MICROSECOND, -123456, '1985/05/02')
FROM iq_dummy
```

The following statement returns the value 2009-11-03 11:10:42.033192:

```
SELECT DATEADD(MCS, 2, '2009-11-03 11:10:42.033190')
FROM iq_dummy
```

Usage

DATEADD is a Transact-SQL compatible data manipulation function.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *DATECEILING Function [Date and Time]* on page 192
- *DATEDIFF Function [Date and Time]* on page 195
- *DATEFLOOR Function [Date and Time]* on page 197
- *DATEPART Function [Date and Time]* on page 203
- *DATENAME Function [Date and Time]* on page 201
- *DATEROUND Function [Date and Time]* on page 204
- *Date Parts* on page 92

DATECEILING Function [Date and Time]

Calculates a new date, time, or datetime value by increasing the provided value up to the nearest larger value of the specified granularity.

Syntax

```
DATECEILING ( date-part, datetime-expression [,multiple -expression])
```

Parameters

Parameter	Description
date-part	The date part to be added to the date.
datetime-expression	The date, time, or date-time expression containing the value you are evaluating.

Parameter	Description
multiple-expression	(Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by the date-part parameter to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 200-microsecond intervals or 10-minute intervals. If multiple-expression evaluates to zero, evaluates to a negative number, is an explicit NULL constant, or is not a valid value for the specified date-part, SAP Sybase IQ generates an error. If multiple-expression evaluates to a NULL, then the function result is NULL.

Examples

This statement returns the value August 13, 2009 10:40.00.000AM:

```
SELECT DATECEILING( MI, 'August 13, 2009, 10:32.00.132AM', 10) FROM
iq_dummy
```

This statement returns the value August 13, 2009 10:32.35.456800 AM:

```
SELECT DATECEILING( US, 'August 13, 2009, 10:32.35.456789AM', 200 )
FROM iq_dummy
```

This statement returns the value August 13, 2009 10:32.35.600000 AM:

```
SELECT DATECEILING( US, 'August 13, 2009, 10:32.35.456789AM',
200000 ) FROM iq_dummy
```

This statement returns the value August 13, 2009 10:32.35.456789 AM:

```
SELECT DATECEILING( US, 'August 13, 2009, 10:32.35.456789AM') FROM
iq_dummy
```

Usage

This function calculates a new date, time, or datetime value by increasing the provided value up to the nearest larger value with the specified granularity. If you include the optional *multiple-expression* parameter, then the function increases the date and time up to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the *multiple-expression* parameter.

The following date parts are not compatible with **DATECEILING**:

- DayofYear
- WeekDay

- CalYearofWeek
- CalWeekofYear
- CalDayofWeek

If you specify a *multiple-expression* for the microsecond, millisecond, second, minute, or hour date parts, SAP Sybase IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of microsecond start from the current second
- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day

For example, if you specify a multiple of two minutes, SAP Sybase IQ applies two-minute intervals starting at the current hour.

For the microsecond, millisecond, second, minute, and hour date parts, specify a *multiple-expression* value that divides evenly into the range of the specified date part:

- For hours, the valid *multiple-expression* values are: 1, 2, 3, 4, 6, 8, 12, 24
- For seconds and minutes, the valid *multiple-expression* values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60
- For milliseconds, the valid *multiple-expression* values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For microseconds, the valid *multiple-expression* values are:

1	40	400	4000	40000
2	50	500	5000	50000
4	64	625	6250	62500
5	80	800	8000	100000
8	100	1000	10000	125000
10	125	1250	12500	200000
16	160	1600	15625	250000
20	200	2000	20000	500000
25	250	2500	25000	1000000
32	320	3125	31250	

If you specify a *multiple-expression* for the day, week, month, quarter, or year date parts, SAP Sybase IQ assumes the intervals started at the smallest date value (0001-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0001-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, SAP Sybase IQ calculates 10-day intervals starting at 0001-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a multiple that divides evenly into the next larger unit of time granularity.

If SAP Sybase IQ rounds to a multiple of the week date part, the date value is always Sunday.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also
- *DATEADD Function [Date and Time]* on page 191
- *DATEDIFF Function [Date and Time]* on page 195
- *DATEFLOOR Function [Date and Time]* on page 197
- *DATEPART Function [Date and Time]* on page 203
- *DATENAME Function [Date and Time]* on page 201
- *DATEROUND Function [Date and Time]* on page 204
- *Date Parts* on page 92

DATEDIFF Function [Date and Time]

Returns the interval between two dates.

Syntax

```
DATEDIFF ( date-part, date-expression1, date-expression2 )
```

Parameters

Parameter	Description
date-part	Specifies the date part in which the interval is to be measured.
date-expression1	The starting date for the interval. This value is subtracted from date-expression2 to return the number of date parts between the two arguments.
date-expression2	The ending date for the interval. *date-expression1* is subtracted from this value to return the number of date parts between the two arguments.

Returns

INT

Examples

The following statement returns 1:

```
SELECT DATEDIFF( HOUR, '4:00AM', '5:50AM' )
FROM iq_dummy
```

The following statement returns 102:

```
SELECT DATEDIFF( MONTH, '1987/05/02', '1995/11/15' )
FROM iq_dummy
```

The following statement returns 0:

```
SELECT DATEDIFF( DAY, '00:00', '23:59' ) FROM iq_dummy
```

The following statement returns 4:

```
SELECT DATEDIFF( DAY, '1999/07/19 00:00', '1999/07/23
23:59' ) FROM iq_dummy
```

The following statement returns 0:

```
SELECT DATEDIFF( MONTH, '1999/07/19', '1999/07/23' )
FROM iq_dummy
```

The following statement returns 1:

```
SELECT DATEDIFF( MONTH, '1999/07/19', '1999/08/23' )
FROM iq_dummy
```

The following statement returns 4:

```
SELECT DATEDIFF(MCS, '2009-11-03 11:10:42.033185',
'2009-11-03 11:10:42.033189') FROM iq_dummy
```

The following statement returns 15:

```
SELECT DATEDIFF(MICROSECOND, '2009-11-10
14:57:52.722001', '2009-11-10 14:57:52.722016')
FROM iq_dummy
```

The following statement returns 1,500,000:

```
SELECT DATEDIFF(MCS, '2000/07/07/07 07:07:06.277777',
'2000/07/07/07 07:07:07.777777') FROM iq_dummy
```

Usage

This function calculates the number of date parts between two specified dates. The result is a signed integer value equal to **(date2 - date1)**, in date parts.

DATEDIFF results are truncated, not rounded, when the result is not an even multiple of the date part.

When you use **day** as the date part, **DATEDIFF** returns the number of midnights between the two times specified, including the second date, but not the first. For example, the following statement returns the value 5. Midnight of the first day 2003/08/03 is not included in the result. Midnight of the second day is included, even though the time specified is before midnight.

```
SELECT DATEDIFF( DAY, '2003/08/03 14:00', '2003/08/08 14:00' ) FROM
iq_dummy
```

When you use **month** as the date part, **DATEDIFF** returns the number of first-of-the-months between two dates, including the second date but not the first. For example, both of the following statements return the value 9:

```
SELECT DATEDIFF( MONTH, '2003/02/01', '2003/11/15' ) FROM iq_dummy;
SELECT DATEDIFF( MONTH, '2003/02/01', '2003/11/01' ) FROM iq_dummy;
```

The first date 2003/02/01 is a first-of-month, but is not included in the result of either query. The second date 2003/11/01 in the second query is also a first-of-month and is included in the result.

When you use **week** as the date part, **DATEDIFF** returns the number of Sundays between the two dates, including the second date but not the first. For example, in the month 2003/08, the dates of the Sundays are 03, 10, 17, 24, and 31. The following query returns the value 4:

```
SELECT DATEDIFF( week, '2003/08/03', '2003/08/31' ) FROM iq_dummy;
```

The first Sunday (2003/08/03) is not included in the result.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *DATEADD Function [Date and Time]* on page 191
- *DATECEILING Function [Date and Time]* on page 192
- *DATEFLOOR Function [Date and Time]* on page 197
- *DATEPART Function [Date and Time]* on page 203
- *DATENAME Function [Date and Time]* on page 201
- *DATEROUND Function [Date and Time]* on page 204
- *Date Parts* on page 92

DATEFLOOR Function [Date and Time]

Calculates a new date, time, or datetime value by reducing the provided value down to the nearest lower value of the specified multiple with the specified granularity.

Syntax

```
DATEFLOOR ( date-part, datetime-expression [,multiple-expression])
```

Parameters

Parameter	Description
date part	The date part to be added to the date.
datetime-expression	The date, time, or date-time expression containing the value you are evaluating.
multiple-expression	(Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by date-part to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 200-microsecond intervals or 10-minute intervals If multiple-expression evaluates to zero, evaluates to a negative number, is an explicit NULL constant, or is not a valid value for the specified date-part, SAP Sybase IQ generates an error. If multiple-expression evaluates to a NULL, then the function result is NULL.

Examples

- This statement returns the value August 13, 2009 10:35:00.000AM:

  ```
  SELECT DATEFLOOR( MINUTE, 'August 13, 2009 10:35:22.123AM') FROM
  iq_dummy
  ```
- This statement returns the value August 13, 2009 10:32:35.456600 AM:

  ```
  SELECT DATEFLOOR( US, 'August 13, 2009, 10:32:35.456789AM', 200 )
  FROM iq_dummy
  ```
- This statement returns the value August 13, 2009 10:32:35.400000 AM:

  ```
  SELECT DATEFLOOR( US, 'August 13, 2009, 10:32:35.456789AM',
  200000 ) FROM iq_dummy
  ```
- This statement returns the value August 13, 2009 10:32:35.456789 AM:

  ```
  SELECT DATEFLOOR( US, 'August 13, 2009, 10:32:35.456789AM') FROM
  iq_dummy
  ```

Usage

This function calculates a new date, time, or datetime value by reducing the provided value down to the nearest lower value with the specified granularity. If you include the optional *multiple-expression* parameter, then the function reduces the date and time down to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the *multiple-expression* parameter.

The following date parts are not compatible with **DATEFLOOR**:

- DayofYear
- WeekDay
- CalYearofWeek
- CalWeekofYear
- CalDayofWeek

If you specify a *multiple-expression* for the microsecond, millisecond, second, minute, or hour date parts, SAP Sybase IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of microsecond start from the current second
- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day

For example, if you specify a multiple of two minutes, SAP Sybase IQ applies two minute intervals starting at the current hour.

For the microsecond, millisecond, second, minute, and hour date parts, specify a *multiple-expression* value that divides evenly into the range of the specified date part:

- For hours, the valid *multiple-expression* values are: 1, 2, 3, 4, 6, 8, 12, 24
- For seconds and minutes, the valid *multiple-expression* values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60
- For milliseconds, the valid *multiple-expression* values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For microseconds, the valid *multiple-expression* values are:

1	40	400	4000	40000
2	50	500	5000	50000
4	64	625	6250	62500
5	80	800	8000	100000
8	100	1000	10000	125000
10	125	1250	12500	200000
16	160	1600	15625	250000
20	200	2000	20000	500000
25	250	2500	25000	1000000

32	320	3125	31250	

If you specify a *multiple-expression* for the day, week, month, quarter, or year date parts, SAP Sybase IQ assumes the intervals started at the smallest date value (0001-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0001-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, then SAP Sybase IQ calculates 10-day intervals starting at 0001-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a multiple that divides evenly into the next larger unit of time granularity.

If SAP Sybase IQ rounds to a multiple of the week date part, the date value is always Sunday.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

- *DATEADD Function [Date and Time]* on page 191
- *DATECEILING Function [Date and Time]* on page 192
- *DATEDIFF Function [Date and Time]* on page 195
- *DATEPART Function [Date and Time]* on page 203
- *DATENAME Function [Date and Time]* on page 201
- *DATEROUND Function [Date and Time]* on page 204
- *Date Parts* on page 92

DATEFORMAT Function [Date and Time]

Returns a string representing a date expression in the specified format.

Syntax

```
DATEFORMAT ( datetime-expression, string-expression )
```

Parameters

Parameter	Description
datetime-expression	The date/time to be converted. Must be a date, time, timestamp, or character string.
string-expression	The format of the converted date.

Returns

VARCHAR

Example

The following statement returns string values like “Jan 01, 1989”:

```
SELECT DATEFORMAT( start_date, 'Mmm dd, yyyy' ) from Employees;
```

The following statement returns the string “Feb 19, 1987”:

```
SELECT DATEFORMAT( CAST ( ‘1987/02/19’ AS DATE ), ‘Mmm Dd, yyyy’ )
FROM iq_dummy
```

Usage

The *datetime-expression* to convert must be a date, time, or timestamp data type, but can also be a `CHAR` or `VARCHAR` character string. If the date is a character string, SAP Sybase IQ implicitly converts the character string to date, time, or timestamp data type, so an explicit cast, as in the example above, is unnecessary.

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

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

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

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

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

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

DATENAME Function [Date and Time]

Returns the name of the specified part (such as the month “June”) of a date/time value, as a character string.

Syntax

```
DATENAME ( date-part, date-expression )
```

Parameters

Table 24. Parameters

Parameter	Description
date-part	The date part to be named.

Parameter	Description
date-expression	The date for which the date part name is to be returned. The date must contain the requested date-part.

Returns
VARCHAR

Example
The following statement returns the value May:

```
SELECT DATENAME( MONTH , '1987/05/02' )
FROM iq_dummy
```

The following statement returns the value 722,001:

```
SELECT DATENAME(MICROSECOND, '2009-11-10
14:57:52.722001') FROM iq_dummy
```

The following statement returns the value 777,777:

```
SELECT DATENAME(MICROSECOND, '2000/07/07
07:07:07.777777') FROM iq_dummy
```

The following statement returns the value 33,189:

```
SELECT DATENAME(MCS, '2009-11-03 11:10:42.033189')
FROM iq_dummy
```

Usage
DATENAME returns a character string, even if the result is numeric, such as 23, for the day.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also
- *DATEADD Function [Date and Time]* on page 191
- *DATECEILING Function [Date and Time]* on page 192
- *DATEDIFF Function [Date and Time]* on page 195
- *DATEFLOOR Function [Date and Time]* on page 197
- *DATEPART Function [Date and Time]* on page 203
- *DATEROUND Function [Date and Time]* on page 204
- *Date Parts* on page 92

DATEPART Function [Date and Time]

Returns an integer value for the specified part of a date/time value.

Syntax

```
DATEPART ( date-part, date-expression )
```

Parameters

Table 25. Parameters

Parameter	Description
date-part	The date part to be returned.
date-expression	The date for which the part is to be returned. The date must contain the date-part field.

Returns

INT

Example

The following statement returns the value 5:

```
SELECT DATEPART( MONTH, '1987/05/02' )
FROM iq_dummy
```

The following statement returns the value 722,001:

```
SELECT DATEPART(MICROSECOND, '2009-11-10
14:57:52.722001') FROM iq_dummy
```

The following statement returns the value 777,777:

```
SELECT DATEPART(MICROSECOND, '2000/07/07
07:07:07.777777') FROM iq_dummy
```

The following statement returns the value 33,189:

```
SELECT DATEPART(MCS, '2009-11-03 11:10:42.033189')
FROM iq_dummy
```

Usage

The **DATE**, **TIME**, and **DTTM** indexes do not support some date parts (Calyearofweek, Calweekofyear, Caldayofweek, Dayofyear, Millisecond, Microsecond).

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *DATEADD Function [Date and Time]* on page 191
- *DATECEILING Function [Date and Time]* on page 192
- *DATEDIFF Function [Date and Time]* on page 195
- *DATEFLOOR Function [Date and Time]* on page 197
- *DATENAME Function [Date and Time]* on page 201
- *DATEROUND Function [Date and Time]* on page 204
- *Date Parts* on page 92

DATEROUND Function [Date and Time]

Calculates a new date, time, or datetime value by rounding the provided value up or down to the nearest multiple of the specified value with the specified granularity.

Syntax

```
DATEROUND (date-part, datetime-expression [,multiple-expression] )
```

Parameters

Parameter	Description
date part	The date part to be returned.
datetime-expression	he date, time, or date-time expression containing the value you are evaluating.
multiple-expression	(Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by date-part to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 200-microsecond intervals or 10-minute intervals. If *multiple-expression* evaluates to zero, evaluates to a negative number, is an explicit NULL constant, or is not a valid value for the specified *date-part*, then SAP Sybase IQ generates an error. If *multiple-expression* evaluates to a NULL, then the function result is NULL.

Examples

This statement returns the value August 13, 2009, 10:30.000AM:

```
SELECT DATEROUND( MI, 'August 13, 2009 10:33.123AM', 10) FROM
iq_dummy
```

This statement returns the value August 13, 2009 10:32:35.456600 AM:

```
SELECT DATEROUND( US, 'August 13, 2009, 10:32:35.456500AM', 200 )
FROM iq_dummy
```

This statement returns the value August 13, 2009 10:32:35.456789 AM:

```
SELECT DATEROUND( US, 'August 13, 2009, 10:32:35.456789AM') FROM
iq_dummy
```

This statement returns the value August 13, 2009 10:32:35.456400 AM:

```
SELECT DATEROUND( US, 'August 13, 2009, 10:32:35.456499AM', 200 )
FROM iq_dummy
```

Usage

This function calculates a new date, time, or datetime value by rounding the provided value up or down to the nearest value with the specified granularity. If you include the optional *multiple-expression* parameter, then the function rounds the date and time to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the *multiple-expression* parameter.

The following date parts are not compatible with **DATEROUND**:

- DayofYear
- WeekDay
- CalYearofWeek
- CalWeekofYear
- CalDayofWeek

If you specify a *multiple-expression* for the microsecond, millisecond, second, minute, or hour date parts, SAP Sybase IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of microsecond start from the current second
- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day

For example, if you specify a multiple of two minutes, SAP Sybase IQ applies two minute intervals starting at the current hour.

For the microsecond, millisecond, second, minute, and hour date parts, specify a *multiple-expression* value that divides evenly into the range of the specified date part:

- For hours, the valid *multiple-expression* values are: 1, 2, 3, 4, 6, 8, 12, 24
- For seconds and minutes, the valid *multiple-expression* values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60

- For milliseconds, the valid *multiple-expression* values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For microseconds, the valid *multiple-expression* values are:

1	40	400	4000	40000
2	50	500	5000	50000
4	64	625	6250	62500
5	80	800	8000	100000
8	100	1000	10000	125000
10	125	1250	12500	200000
16	160	1600	15625	250000
20	200	2000	20000	500000
25	250	2500	25000	1000000
32	320	3125	31250	

If you specify a *multiple-expression* for the day, week, month, quarter, or year date parts, SAP Sybase IQ assumes the intervals started at the smallest date value (0001-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0001-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, then SAP Sybase IQ calculates 10-day intervals starting at 0001-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a multiple that divides evenly into the next larger unit of time granularity.

If SAP Sybase IQ rounds to a multiple of the week date part, then the date value is always Sunday.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

- *DATEADD Function [Date and Time]* on page 191
- *DATECEILING Function [Date and Time]* on page 192
- *DATEDIFF Function [Date and Time]* on page 195
- *DATEFLOOR Function [Date and Time]* on page 197
- *DATEPART Function [Date and Time]* on page 203
- *DATENAME Function [Date and Time]* on page 201
- *Date Parts* on page 92

DATETIME Function [Date and Time]

Converts an expression into a timestamp.

Syntax

```
DATETIME ( expression )
```

Parameters

Parameter	Description
expression	The expression to be converted. The expression is usually a string. Conversion errors may be reported.

Returns

TIMESTAMP

Example

The following statement returns a timestamp with value 1998-09-09 12:12:12.000:

```
SELECT DATETIME( '1998-09-09 12:12:12.000' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

DAY Function [Date and Time]

Returns an integer from 1 to 31 corresponding to the day of the month of the date specified.

Syntax

```
DAY ( date-expression )
```

Parameters

Table 26. Parameters

Parameter	Description
date-expression	The date.

Returns

SMALLINT

Example

The following statement returns the value 12:

```
SELECT DAY( '2001-09-12' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

DAYNAME Function [Date and Time]

Returns the name of the day of the week from the specified date.

Syntax

```
DAYNAME ( date-expression )
```

Parameters

Table 27. Parameters

Parameter	Description
date-expression	The date.

Returns

VARCHAR

Example

The following statement returns the value Saturday:

```
SELECT DAYNAME ( '1987/05/02' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

DAYS Function [Date and Time]

Returns the number of days since an arbitrary starting date, returns the number of days between two specified dates, or adds the specified *integer-expression* number of days to a given date.

DAYS ignores hours, minutes, and seconds.

Syntax

```
DAYS ( datetime-expression )
|   ( datetime-expression, datetime-expression )
|   ( datetime-expression, integer-expression )
```

Parameters

Table 28.

Parameter	Description
datetime-expression	A date and time.
integer-expression	The number of days to be added to the datetime-expression. If the integer-expression is negative, the appropriate number of days are subtracted from the date/time. If you supply an integer expression, the datetime-expression must be explicitly cast as a date.

Returns

INT when you specify two datetime expressions.

TIMESTAMP when the second argument you specify is an integer.

Examples

The following statement returns the integer value 729948:

```
SELECT DAYS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the integer value -366, which is the difference between the two dates:

```
SELECT DAYS( '1998-07-13 06:07:12',
'1997-07-12 10:07:12' ) FROM iq_dummy
```

The following statement returns the value 1999-07-14:

```
SELECT DAYS( CAST('1998-07-13' AS DATE ), 366 )
FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *CAST Function [Data Type Conversion]* on page 167

DB_ID Function [System]

Returns the database ID number.

Syntax

```
DB_ID ( [ database-name ] )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
database-name	A string expression containing the database name. If database-name is a string constant, it must be enclosed in quotes. If no database-name is supplied, the ID number of the current database is returned

Returns

INT

Examples

Returns the value 0, if `iqdemo` is the only running database:

```
SELECT DB_ID( 'iqdemo' ) FROM iq_dummy
```

Returns the value 0, if executed against the only running database:

```
SELECT DB_ID() FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ.

See also

- *COL_NAME Function [System]* on page 175
- *DB_NAME Function [System]* on page 211
- *DB_PROPERTY Function [System]* on page 212
- *NEXT_DATABASE Function [System]* on page 283
- *OBJECT_ID Function [System]* on page 290
- *OBJECT_NAME Function [System]* on page 291

DB_NAME Function [System]

Returns the database name.

Syntax

```
DB_NAME ( [ database-id ] )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
database-id	The ID of the database. database-id must be a numeric expression

Returns

VARCHAR

Example

Returns the database name **iqdemo**, when executed against the demo database:

```
SELECT DB_NAME( 0 ) FROM iq_dummy
```

Usage

If no *database-id* is supplied, the name of the current database is returned.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ.

See also

- *COL_NAME Function [System]* on page 175
- *DB_ID Function [System]* on page 210
- *DB_PROPERTY Function [System]* on page 212
- *NEXT_DATABASE Function [System]* on page 283
- *OBJECT_ID Function [System]* on page 290
- *OBJECT_NAME Function [System]* on page 291

DB_PROPERTY Function [System]

Returns the value of the given property.

Syntax

```
DB_PROPERTY ( { property-id | property-name }
[ , { database-id | database-name } ] )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Table 29. Parameters

Parameter	Description
property-id	The database property ID.
property-name	The database property name.
database-id	The database ID number, as returned by DB_ID. Typically, the database name is used.
database-name	The name of the database, as returned by DB_NAME.

Returns

VARCHAR

Example

The following statement returns the page size of the current database, in bytes.

```
SELECT DB_PROPERTY( 'PAGESIZE' ) FROM iq_dummy
```

Usage

Returns a string. The current database is used if the second argument is omitted.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *COL_NAME Function [System]* on page 175
- *DB_ID Function [System]* on page 210
- *DB_NAME Function [System]* on page 211
- *NEXT_DATABASE Function [System]* on page 283

- *OBJECT_ID Function [System]* on page 290
- *OBJECT_NAME Function [System]* on page 291

DEGREES Function [Numeric]

Converts a number from radians to degrees.

Syntax

```
DEGREES ( numeric-expression )
```

Parameters

Table 30. Parameters

Parameter	Description
numeric-expression	An angle in radians.

Returns

Returns the degrees of the angle given by *numeric-expression*.

DOUBLE

Example

The following statement returns the value 29.793805:

```
SELECT DEGREES( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

DENSE_RANK Function [Analytical]

Ranks items in a group.

Syntax

```
DENSE_RANK () OVER ( ORDER BY expression [ ASC | DESC ] )
```

Parameters

Table 31. Parameters

Parameter	Description
expression	A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Returns
INTEGER

Example
The following statement illustrates the use of the **DENSE_RANK** function:

```
SELECT s_suppkey, DENSE_RANK()
OVER ( ORDER BY ( SUM(s_acctBal) DESC )
AS rank_dense FROM supplier GROUP BY s_suppkey;

s_suppkey        sum_acctBal       rank_dense
supplier#011     200,000            1
supplier#002     200,000            1
supplier#013     123,000            2
supplier#004     110,000            3
supplier#035     110,000            3
supplier#006     50,000             4
supplier#021     10,000             5
```

Usage
DENSE_RANK is a rank analytical function. The dense rank of row R is defined as the number of rows preceding and including R that are distinct within the groups specified in the **OVER** clause or distinct over the entire result set. The difference between **DENSE_RANK** and **RANK** is that **DENSE_RANK** leaves no gap in the ranking sequence when there is a tie. **RANK** leaves a gap when there is a tie.

DENSE_RANK requires an **OVER (ORDER BY)** clause. The **ORDER BY** clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the **OVER** clause and is not an **ORDER BY** for the **SELECT**. No aggregation functions in the rank query are allowed to specify **DISTINCT**.

The **OVER** clause indicates that the function operates on a query result set. The result set is the rows that are returned after the **FROM**, **WHERE**, **GROUP BY**, and **HAVING** clauses have all been evaluated. The **OVER** clause defines the data set of the rows to include in the computation of the rank analytical function.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

DENSE_RANK is allowed only in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. **DENSE_RANK** can be in a view or a union. The **DENSE_RANK** function cannot be used in a subquery, a HAVING clause, or in the select list of an **UPDATE** or **DELETE** statement. Only one rank analytical function is allowed per query.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

- *RANK Function [Analytical]* on page 307

DIFFERENCE Function [String]

Compares two strings, evaluates the similarity between them, and returns a value from 0 to 4.

The best match is 4.

Syntax

```
DIFFERENCE ( string-expression1, string-expression2 )
```

Parameters

Table 32. Parameters

Parameter	Description
string-expression1	The first string to compare.
string-expression2	The second string to compare.

Returns
SMALLINT

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *SOUNDEX Function [String]* on page 340

DIFFERENCE Function Examples

Use the examples as reference for **DIFFERENCE** function usage.

The following statement returns the value 4:

```
SELECT DIFFERENCE( 'Smith', 'Smith' ) FROM iq_dummy
```

The following statement returns the value 4:

```
SELECT DIFFERENCE( 'Smith', 'Smyth' ) FROM iq_dummy
```

The following statement returns the value 3:

```
SELECT DIFFERENCE( 'Smith', 'Sweeney' ) FROM iq_dummy
```

The following statement returns the value 2:

```
SELECT DIFFERENCE( 'Smith', 'Jones' ) FROM iq_dummy
```

The following statement returns the value 1:

```
SELECT DIFFERENCE( 'Smith', 'Rubin' ) FROM iq_dummy
```

The following statement returns the value 0:

```
SELECT DIFFERENCE( 'Smith', 'Wilkins' ) FROM iq_dummy
```

DOW Function [Date and Time]

Returns a number from 1 to 7 representing the day of the week of the specified date, with Sunday=1, Monday=2, and so on.

Syntax

```
DOW ( date-expression )
```

Parameters

Table 33. Parameters

Parameter	Description
date-expression	The date.

Returns

SMALLINT

Example

The following statement returns the value 5:

```
SELECT DOW( '1998-07-09' ) FROM iq_dummy
```

Usage

Use the DATE_FIRST_DAY_OF_WEEK option if you need Monday (or another day) to be the first day of the week.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

ERRORMSG Function [Miscellaneous]

Provides the error message for the current error, or for a specified SQLSTATE or SQLCODE value.

Syntax

```
ERRORMSG ( [ sqlstate | sqlcode ] )
```

```
sqlstate: string
```

```
sqlcode: integer
```

Parameters

Parameter	Definition
sqlstate	The SQLSTATE value for which the error message is to be returned.
sqlcode	The SQLCODE value for which the error message is to be returned.

Returns

A string containing the error message.

VARCHAR

Example

The following statement returns the error message for SQLCODE -813:

```
select errormsg( -813 )
```

Return Value

A string containing the error message. If no argument is supplied, the error message for the current state is supplied. Any substitutions (such as table names and column names) are made.

If an argument is supplied, the error message for the supplied SQLSTATE or SQLCODE is returned, with no substitutions. Table names and column names are supplied as placeholders ('???').

The **ERRORMSG** function returns SQL Anywhere and SAP Sybase IQ error messages.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

EVENT_CONDITION Function [System]

Specifies when an event handler is triggered.

To define an event and its associated handler, use the **CREATE EVENT** statement.

Syntax

```
EVENT_CONDITION ( condition-name )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Table 34. Parameters

Parameter	Definition
condition-name	The condition triggering the event. The possible values are preset in the database, and are case-insensitive. Each condition is valid only for certain event types.

Table 35. Valid conditions for events

Condition name	Units	Valid for	Comment
DBFreePercent	N/A	DBDiskSpace	DBDiskSpace shows free space in the system database file (.db file), not the IQ store.
DBFreeSpace	Megabytes	DBDiskSpace	
DBSize	Megabytes	GrowDB	
ErrorNumber	N/A	RAISERROR	
IdleTime	Seconds	ServerIdle	
Interval	Seconds	All	Time since handler last executed.

Condition name	Units	Valid for	Comment
LogFreePercent	N/A	LogDiskSpace	
LogFreeSpace	Megabytes	LogDiskSpace	
LogSize	Megabytes	GrowLog	
RemainingValues	Integer	GlobalAutoincrement	The number of remaining values.
TempFreePercent	N/A	TempDiskSpace	TempDiskSpace shows free space in the system temporary file (pointed to by TEMP or IQTMP16 environment variable), not the IQ temporary store.
TempFreeSpace	Megabytes	TempDiskSpace	
TempSize	Megabytes	GrowTemp	

Returns
INT

Example
The following event definition uses the **EVENT_CONDITION** function:

```
create event LogNotifier
type LogDiskSpace
where event_condition( 'LogFreePercent' ) < 50
handler
begin
    message 'LogNotifier message'
end
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also
- *EVENT_PARAMETER Function [System]* on page 220

EVENT_CONDITION_NAME Function [System]

Can be used to list the possible parameters for **EVENT_CONDITION**.

To define an event and its associated handler, use the **CREATE EVENT** statement.

Syntax

```
EVENT_CONDITION_NAME ( integer )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
integer	Must be greater than or equal to zero.

Returns

VARCHAR

Usage

You can use **EVENT_CONDITION_NAME** to obtain a list of all **EVENT_CONDITION** arguments by looping over integers until the function returns NULL.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

EVENT_PARAMETER Function [System]

Provides context information for event handlers.

To define an event and its associated handler, use the **CREATE EVENT** statement.

Syntax

```
EVENT_PARAMETER ( context-name )
```

```
context-name:
  'ConnectionID'
|  'User'
|  'EventName'
|  'Executions'
|  'IQDBMainSpaceName'
|  'NumActive'
|  'TableName'
|  condition-name
```

Note: CIS functional compensation performance considerations apply.

Parameters

Table 36. Parameters

Parameter	Description
context-name	One of the preset strings. The strings are case-insensitive, and carry the following information:
ConnectionId	The connection ID, as returned by `connection_property( 'id' )`
User	The user ID for the user that caused the event to be triggered.
EventName	The name of the event that has been triggered.
Executions	The number of times the event handler has been executed.
NumActive	The number of active instances of an event handler. This is useful if you want to limit an event handler so that only one instance executes at any given time.
TableName	The name of the table, for use with RemainingValues.

In addition, you can access any of the valid *condition-name* arguments to the **EVENT_CONDITION** function from the **EVENT_PARAMETER** function.

Returns
VARCHAR

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *EVENT_CONDITION Function [System]* on page 218

EXP Function [Numeric]

Returns the exponential function, e to the power of a number.

Syntax

```
EXP ( numeric-expression )
```

Parameters

Table 37. Parameters

Parameter	Description
numeric-expression	The exponent.

Returns

DOUBLE

Example

The following statement returns the value 3269017.3724721107:

```
SELECT EXP( 15 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

EXP_WEIGHTED_AVG Function [Aggregate]

Calculates an exponential weighted moving average.

Weightings determine the relative importance of each quantity that makes up the average.

Syntax

```
EXP_WEIGHTED_AVG (expression, period-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
expression	A numeric expression for which a weighted value is being computed.
period-expression	A numeric expression specifying the period for which the average is to be computed.

Usage

Similar to the **WEIGHTED_AVG** function, the weights in **EXP_WEIGHTED_AVG** decrease over time. However, weights in WEIGHTED_AVG decrease arithmetically, whereas weights in EXP_WEIGHTED_AVG decrease exponentially. Exponential weighting applies more weight to the most recent values, and decreases the weight for older values while still applying some weight.

SAP Sybase IQ calculates the exponential moving average using:

S*C+(1-S)*PEMA

In the calculation above, SAP Sybase IQ applies the smoothing factor by multiplying the current closing price (C) by the smoothing constant (S) added to the product of the previous day's exponential moving average value (PEMA) and 1 minus the smoothing factor.

SAP Sybase IQ calculates the exponential moving average over the entire period specified by the **OVER** clause. *period-expression* specifies the moving range of the exponential moving average.

You can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement. The *window-spec* must contain an **ORDER BY** statement and cannot contain a frame specification.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause. DISTINCT is not supported.

Example

The following example returns an exponential weighted average of salaries for employees in Florida with the salary of recently hired employees contributing the most weight to the average. There are three rows used in the weighting:

```
SELECT DepartmentID, Surname, Salary,EXP_WEIGHTED_AVG(Salary, 3)
OVER (ORDER BY YEAR(StartDate) DESC) as "W_AVG"FROM EmployeesWHERE
State IN ('FL') ORDER BY StartDate DESC
```

The returned result set is:

Table 38. EXP_WEIGHTED_AVG result set

DepartmentID	Surname	Salary	W_AVG
400	Evans	68,940.000	34,470.000000
300	Litton	58,930.000	46,700.000000
200	Sterling	64,900.000	55,800.000000
200	Kelly	87,500.000	71,650.000000
400	Charlton	28,300.000	49,975.000000

DepartmentID	Surname	Salary	W_AVG
100	Lull	87,900.000	68,937.500000
100	Gowda	59,840.000	60,621.875000
400	Francis	53,870.000	61,403.750000

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

See also

- *WEIGHTED_AVG Function [Aggregate]* on page 376
- *Windowing Aggregate Function Usage* on page 83

FIRST_VALUE Function [Aggregate]

Returns the first value from a set of values.

Syntax

```
FIRST_VALUE (expression [IGNORE NULLS | RESPECT NULLS])

OVER (window-spec)
```

Parameters

Parameter	Description
expression	The expression on which to determine the first value in an ordered set.

Returns

Data type of the argument.

Usage

FIRST_VALUE returns the first value in a set of values, which is usually an ordered set. If the first value in the set is null, then the function returns NULL unless you specify **IGNORE NULLS**. If you specify IGNORE NULLS, then **FIRST_VALUE** returns the first non-null value in the set, or NULL if all values are null.

The data type of the returned value is the same as that of the input value.

You cannot use **FIRST_VALUE** or any other analytic function for expression. That is, you cannot nest analytic functions, but you can use other built-in function expressions for expression.

If the window-spec does not contain an **ORDER BY** expression, or if the **ORDER BY** expression is not precise enough to guarantee a unique ordering, then the result is arbitrary. If there is no window-spec, then the result is arbitrary.

You can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Note: DISTINCT is not supported.

Example

The following example returns the relationship, expressed as a percentage, between each employee's salary and that of the most recently hired employee in the same department:

```
SELECT DepartmentID, EmployeeID,
100 * Salary / ( FIRST_VALUE( Salary ) OVER (
PARTITION BY DepartmentID  ORDER BY Year(StartDate) DESC ) )
AS percentage
FROM Employees order by DepartmentID DESC;
```

The returned result set is:

Table 39. FIRST_VALUE result set

DepartmentID	EmployeeID	Percentage
500	1,658	100.000000000000000000000
500	1,570	138.842709713689113761394
500	1,615	110.428462434244870095972
500	1,013	109.585190539292454724330
500	750	137.734409508894510701521
500	921	167.449704854836766654619
500	868	113.239368750752921334778
500	703	222.867927558928643135365
500	191	119.664297474199895594908
400	1,684	100.000000000000000000000
400	1,740	76.128652163477274215016
400	1,751	76.353400685155687446813
400	1,607	133.758100765890593292456
400	1,507	77.996465120338650199655

DepartmentID	EmployeeID	Percentage
400	1,576	150.428767810774836893669

In this example, employee 1658 is the first row for department 500, indicating that employee 1658 is the most recent hire in that department, and therefore receives a percentage of 100%. Percentages for the remaining employees in department 500 are calculated relative to that of employee 1658. For example, employee 1570 earns approximately 139% of what employee 1658 earns.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

See also

- *Windowing Aggregate Function Usage* on page 83

FLOOR Function [Numeric]

Returns the floor of (largest integer not greater than) a number.

Syntax

```
FLOOR ( numeric-expression )
```

Parameters

Table 40. Parameters

Parameter	Description
numeric-expression	The number, usually a float.

Returns

DOUBLE

Examples

The following statement returns the value 123.00:

```
SELECT FLOOR ( 123 ) FROM iq_dummy
```

The following statement returns the value 123:

```
SELECT FLOOR ( 123.45 ) FROM iq_dummy
```

The following statement returns the value -124.00.

```
SELECT FLOOR ( -123.45 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *CEILING Function [Numeric]* on page 169
- *CEIL Function [Numeric]* on page 169

GETDATE Function [Date and Time]

Returns the current date and time.

Syntax

```
GETDATE ()
```

Returns

TIMESTAMP

Example

The following statement returns the system date and time.

```
SELECT GETDATE( ) FROM iq_dummy
```

Usage

GETDATE is a Transact-SQL compatible data manipulation function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

GRAPHICAL_PLAN Function [String]

Returns the graphical query plan to Interactive SQL in an XML format string.

Syntax

```
GRAPHICAL_PLAN ( string-expression
[, statistics-level
[, cursor-type
[, update-status ]]])
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
string-expression	SQL statement for which the plan is to be generated. string-expression is generally a **SELECT** statement, but it can also be an **UPDATE** or **DELETE**, **INSERT SELECT**, or **SELECT INTO** statement.
statistics-level	An integer. Statistics-level can be: • 0 – Optimizer estimates only (default). • 2 – Detailed statistics including node statistics. • 3 – Detailed statistics.
cursor-type	A cursor type, expressed as a string. Possible values are: asensitive, insensitive, sensitive, or keyset-driven. If cursor-type is not specified, asensitive is used by default.
update-status	A string parameter accepting one of the following values indicating how the optimizer should treat the given cursor: READ-ONLY – The cursor is read-only. READ-WRITE (default) – The cursor can be read or written to. READ-WRITE (default) – The cursor can be read or written to.

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **GRAPHICAL_PLAN** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **GRAPHICAL_PLAN** to the correct data type and size.

Usage

If you do not provide an argument to the **GRAPHICAL_PLAN** function, the query plan is returned to you from the cache. If there is no query plan in the cache, then this message appears:

```
plan not available
```

The behavior of **GRAPHICAL_PLAN** function is controlled by database options `QUERY_PLAN_TEXT_ACCESS` and `QUERY_PLAN_TEXT_CACHING`. If `QUERY_PLAN_TEXT_ACCESS` is OFF (the default), then this message appears:

```
Plan not available. The database option QUERY_PLAN_TEXT_ACCESS is OFF
```

If a user needs access to the plan, a user with the SET ANY SYSTEM OPTION system privilege must set option `QUERY_PLAN_TEXT_ACCESS` ON for that user.

If `QUERY_PLAN_TEXT_ACCESS` is ON, and the query plan for the string expression is available in the cache maintained on the server, the query plan from the cache is returned to you.

If the query plan is not available in the cache and you are authorized to view plans on the client, then a query plan with optimizer estimates (query plan with `NOEXEC` option ON) is generated and appears on the Interactive SQL client plan window.

When a user requests a query plan that has not yet been executed, the query plan is not available in the cache. Instead, a query plan with optimizer estimates is returned without `QUERY_PLAN_AFTER_RUN` statistics.

Query plans for stored procedures are not accessible using the **GRAPHICAL_PLAN** function.

Users can view the query plan for cursors opened for SAP Sybase IQ queries. A cursor is declared and opened using **DECLARE CURSOR** and **OPEN CURSOR**. To obtain the query plan for the most recently opened cursor, use:

```
SELECT GRAPHICAL_PLAN ( );
```

With the `QUERY_PLAN_AFTER_RUN` option OFF, the plan appears after **OPEN CURSOR** or **CLOSE CURSOR**. However, if `QUERY_PLAN_AFTER_RUN` is ON, **CLOSE CURSOR** must be executed before you request the plan.

Examples

The following example passes a **SELECT** statement as a string parameter and returns the plan for executing the query. It saves the plan in the file `gplan.xml`.

Note: If you use the **OUTPUT** statement's **HEXADECIMAL** clause set to **ASIS** to get formatted plan output, the values of characters are written without any escaping, even if the value contains control characters. **ASIS** is useful for text that contains formatting characters such as tabs or carriage returns.

```
SELECT GRAPHICAL_PLAN ('SELECT * FROM Employees');OUTPUT to 'C:
\gplan.xml' HEXADECIMAL ASIS quote '';
```

The following example returns the query plan from the cache, if available:

```
SELECT GRAPHICAL_PLAN ( );
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *HTML_PLAN Function [String]* on page 239

GROUPING Function [Aggregate]

Identifies whether a column in a **ROLLUP** or **CUBE** operation result set is NULL because it is part of a subtotal row, or NULL because of the underlying data.

Syntax

```
GROUPING ( group-by-expression )
```

Parameters

Parameter	Description
group-by-expression	An expression appearing as a grouping column in the result set of a query that uses a GROUP BY clause with the ROLLUP or CUBE keyword. The function identifies subtotal rows added to the result set by a ROLLUP or CUBE operation.

Currently, SAP Sybase IQ does not support the **PERCENTILE_CONT** or **PERCENTILE_DISC** functions with **GROUP BY CUBE** operations.

Returns

Value	Description
1	Indicates that group-by-expression is NULL because it is part of a subtotal row. The column is not a prefix column for that row.
0	Indicates that group-by-expression is a prefix column of a subtotal row.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *Windowing Aggregate Function Usage* on page 83

GROUP_MEMBER Function [System]

Identifies whether the user belongs to the specified group.

Syntax

```
GROUP_MEMBER ( group-name-string-expression[ , user-name-string-
expression ] )
```

Parameters

Parameter	Description
group-name-string-expression	Identifies the group to be considered.
user-name-string-expression	Identifies the user to be considered. If not supplied, then the current user name is assumed.

Return Values

Table 41. Return Values

Value	Description
0	Returns 0 if the group does not exist, if the user does not exist, or if the user does not belong to the specified group.
1	Returns an integer other than 0 if the user is a member of the specified group.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

HEXTOBIGINT Function [Data Type Conversion]

Returns the `BIGINT` equivalent of a hexadecimal string.

Syntax

```
HEXTOBIGINT ( hexadecimal-string )
```

Parameters

Parameter	Description

hexadecimal-string	The hexadecimal value to be converted to a big integer (`BIGINT`). Input can be in the following forms, with either a lowercase or uppercase "0x" in the prefix, or no prefix:
	`0xhex-string`
	`0Xhex-string`
	`hex-string`

Examples

The following statements return the value 4294967287:

```
SELECT HEXTOBIGINT ( '0xfffffff7' ) FROM iq_dummy
```

```
SELECT HEXTOBIGINT ( '0Xfffffff7' ) FROM iq_dummy
```

```
SELECT HEXTOBIGINT ( 'fffffff7' ) FROM iq_dummy
```

Usage

The **HEXTOBIGINT** function accepts hexadecimal integers and returns the `BIGINT` equivalent. Hexadecimal integers can be provided as `CHAR` and `VARCHAR` value expressions, as well as `BINARY` and `VARBINARY` expressions.

The **HEXTOBIGINT** function accepts a valid hexadecimal string, with or without a "0x" or "0X" prefix, enclosed in single quotes.

Input of fewer than 16 digits is assumed to be left-padded with zeros.

For data type conversion failure on input, an error is returned unless the `CONVERSION_ERROR` option is set to OFF. When `CONVERSION_ERROR` is OFF, invalid hexadecimal input returns NULL.

See Reference: Statements and Options > Database Options > Alphabetical List of Options > CONVERSION_ERROR Option [TSQL].

An error is returned if a `BINARY` or `VARBINARY` value exceeds 8 bytes and a `CHAR` or `VARCHAR` value exceeds 16 characters, with the exception of the value being appended with '0x.'

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *BIGINTTOHEX Function [Data Type Conversion]* on page 164
- *HEXTOINT Function [Data Type Conversion]* on page 233
- *INTTOHEX Function [Data Type Conversion]* on page 248

HEXTOINT Function [Data Type Conversion]

Returns the unsigned `BIGINT` equivalent of a hexadecimal string.

Syntax

```
HEXTOINT ( hexadecimal-string )
```

Parameters

Parameters	Description

hexadecimal-string	The string to be converted to an integer. Input can be in the following forms, with either a lowercase or uppercase “x” in the prefix, or no prefix:
	`0xhex-string`
	`0Xhex-string`
	`hex-string`

Returns

The HEXTOINT function returns the platform-independent SQL INTEGER equivalent of the hexadecimal string. The hexadecimal value represents a negative integer if the 8th digit from the right is one of the digits 8-9 and the uppercase or lowercase letters A-F and the previous leading digits are all uppercase or lowercase letter F. The following is not a valid use of HEXTOINT since the argument represents a positive integer value that cannot be represented as a signed 32-bit integer:

```
SELECT HEXTOINT( '0x0080000001' );
```

INT

Examples

The following statements return the value 420:

```
SELECT HEXTOINT ( '0x1A4' ) FROM iq_dummy
```

```
SELECT HEXTOINT ( '0X1A4' ) FROM iq_dummy
```

```
SELECT HEXTOINT ( '1A4' ) FROM iq_dummy
```

Usage

For invalid hexadecimal input, SAP Sybase IQ returns an error unless the `CONVERSION_ERROR` option is OFF. When `CONVERSION_ERROR` is OFF, invalid hexadecimal input returns NULL.

See *CONVERSION_ERROR Option [TSQL]* in *Reference: Statements and Options*.

The database option `ASE_FUNCTION_BEHAVIOR` specifies that output of SAP Sybase IQ functions, including **INTTOHEX** and **HEXTOINT**, is consistent with the output of Adaptive Server Enterprise functions.

See *ASE_FUNCTION_BEHAVIOR Option* in *Reference: Statements and Options*.

When the `ASE_FUNCTION_BEHAVIOR` option is ON:

- SAP Sybase IQ **HEXTOINT** assumes input is a hexadecimal string of 8 characters; if the length is less than 8 characters long, the string is left padded with zeros.
- SAP Sybase IQ **HEXTOINT** accepts a maximum of 16 characters prefixed with 0x (a total of 18 characters); use caution, as a large input value can result in an integer value that overflows the 32-bit signed integer output size.
- The data type of the output of the SAP Sybase IQ **HEXTOINT** function is assumed to be a 32-bit signed integer.
- SAP Sybase IQ **HEXTOINT** accepts a 32-bit hexadecimal integer as a signed representation.
- For more than 8 hexadecimal characters, SAP Sybase IQ **HEXTOINT** considers only relevant characters.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *BIGINTTOHEX Function [Data Type Conversion]* on page 164
- *HEXTOBIGINT Function [Data Type Conversion]* on page 231
- *INTTOHEX Function [Data Type Conversion]* on page 248

HOUR Function [Date and Time]

Returns a number from 0 to 23 corresponding to the hour component of the specified date/time.

Syntax

```
HOUR ( datetime-expression )
```

Table 42. Parameters

Parameter	Definition
datetime-expression	The date/time.

Returns
SMALLINT

Example
The following statement returns the value 21:

```
SELECT HOUR( '1998-07-09 21:12:13' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

HOURS Function [Date and Time]

Returns the number of hours since an arbitrary starting date and time, the number of whole hours between two specified times, or adds the specified integer-expression number of hours to a time.

Syntax

```
HOURS ( datetime-expression
| datetime-expression, datetime-expression
| datetime-expression, integer-expression )
```

Parameters

Table 43. Parameters

Parameter	Description
datetime-expression	A date and time.
integer-expression	The number of hours to be added to the *datetime-expression.* If *integer-expression* is negative, the appropriate number of hours are subtracted from the date/time. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a datetime data type.

Returns
INT

Examples

The following statement returns the value 17518758:

```
SELECT HOURS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 4, to signify the difference between the two times:

```
SELECT HOURS( '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-05-13 02:05:07.000:

```
SELECT HOURS( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5 ) FROM iq_dummy
```

Usage

The second syntax returns the number of whole hours from the first date/time to the second date/time. The number might be negative.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *CAST Function [Data Type Conversion]* on page 167
- *CONVERT Function [Data Type Conversion]* on page 178
- *MINUTES Function [Date and Time]* on page 275
- *MONTHS Function [Date and Time]* on page 278
- *REPLACE Function [String]* on page 321
- *SECOND Function [Date and Time]* on page 331
- *WEEKS Function [Date and Time]* on page 375
- *YEAR Function [Date and Time]* on page 380
- *YEARS Function [Date and Time]* on page 381

HTML_DECODE function [Miscellaneous]

Decodes special character entities that appear in HTML literal strings.

Syntax

```
HTML_DECODE( string )
```

Parameters

- ***string* –** Arbitrary literal string used in an HTML document.

Returns

LONG VARCHAR or LONG NVARCHAR.

Note: The result data type is a LONG VARCHAR. If you use HTML_DECODE in a SELECT INTO statement, you must have an Unstructured Data Analytics Option license or use CAST and set HTML_DECODE to the correct data type and size.

Remarks

This function returns the string argument after making the appropriate substitutions. The following table contains a sampling of the acceptable character entities.

Characters	Substitution
&quot;	"
&#39;	'
&amp;	&
&lt;	<
&gt;	>
&#x*hexadecimal-number*;	Unicode codepoint, specified as a hexadecimal number. For example, &#x27; returns a single apostrophe.
&#*decimal-number*;	Unicode codepoint, specified as a decimal number. For example, &#8482; returns the trademark symbol.

When a Unicode codepoint is specified, if the value can be converted to a character in the database character set, it is converted to a character. Otherwise, it is returned uninterpreted.

SAP Sybase IQ supports all character entity references specified in the HTML 4.01 Specification. See *http://www.w3.org/TR/html4/* and *http://www.w3.org/TR/html4/sgml/entities.html#h-24.2*.

Standards and compatibility

- **SQL/2008 –** Vendor extension.

Examples

```
SELECT HTML_DECODE('&lt;p&gt;The piano was made ' ||
  'by &lsquo;Steinway &amp; Sons&rsquo;.&lt;/p&gt;')
```

```
SELECT HTML_DECODE('&lt;p&gt;It cost &euro;85.000,00.&lt;/p&gt;')
```

HTML_ENCODE function [Miscellaneous]

Encodes special characters within strings to be inserted into HTML documents.

Syntax

```
HTML_ENCODE( string )
```

Parameters

- ***string* –** Arbitrary string to be used in an HTML document.

Returns

LONG VARCHAR or LONG NVARCHAR.

Note: The result data type is a LONG VARCHAR. If you use HTML_ENCODE in a SELECT INTO statement, you must have an Unstructured Data Analytics Option license or use CAST and set HTML_DECODE to the correct data type and size.

Remarks

This function returns the string argument after making the following set of substitutions:

Characters	Substitution
"	&quot;
'	&#39;
&	&amp;
<	&lt;
>	&gt;
codes *nn* less than 0x20	&#x*nn*;

This function supports NCHAR inputs and/or outputs.

Standards and compatibility

- **SQL/2008 –** Vendor extension.

Examples

The following example returns the string '&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;&gt; '.

```
SELECT HTML_ENCODE('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//
EN">')
```

HTML_PLAN Function [String]

Returns query plans in an HTML format string.

Syntax

```
HTML_PLAN ( string-expression )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
string-expression	SQL statement for which the plan is to be generated. It is primarily a **SELECT** statement but can be an **UPDATE** or **DELETE** statement.

If you do not provide an argument to the **HTML_PLAN** function, the query plan is returned to you from the cache. If there is no query plan in the cache, this message appears:

```
No plan available
```

The behavior of the **HTML_PLAN** function is controlled by database options `QUERY_PLAN_TEXT_ACCESS` and `QUERY_PLAN_TEXT_CACHING`. If `QUERY_PLAN_TEXT_ACCESS` is OFF (the default), this message appears:

```
Plan not available. The database option QUERY_PLAN_TEXT_ACCESS is OFF
```

If `QUERY_PLAN_TEXT_ACCESS` is ON, and the query plan for the string expression is available in the cache maintained on the server, the query plan from the cache is returned to you.

The **HTML_PLAN** function can be used to return query plans to Interactive SQL using **SELECT**, **UPDATE**, **DELETE**, **INSERT SELECT**, and **SELECT INTO**.

Users can view the query plan for cursors opened for SAP Sybase IQ queries. To obtain the query plan for the most recently opened cursor, use:

```
SELECT HTML_PLAN ( );
```

With `QUERY_PLAN_AFTER_RUN` option OFF, the plan appears after **OPEN CURSOR** or **CLOSE CURSOR**. However, if `QUERY_PLAN_AFTER_RUN` is ON, **CLOSE CURSOR** must be executed before you request the plan.

When you request an **HTML_PLAN** for a SQL Anywhere query or for an OMNI/CIS decomposed query, the following message is returned:

```
No plan. HTML_PLAN function is not supported for this type of
statement or database.
```

Examples

The following example passes a **SELECT** statement as a string parameter and returns the HTML plan for executing the query. It saves the plan in the file `hplan.html`.

```
SELECT HTML_PLAN ('SELECT * FROM Employees');
OUTPUT to 'C:\hplan.html' HEXADECIMAL ASIS QUOTE '';
```

The **OUTPUT TO** clause **HEXADECIMAL ASIS** is useful for text that contains formatting characters such as tabs or carriage returns. When set to **ASIS**, values are written as is, without any escaping, even if the values contain control characters.

The following example returns the HTML query plan from the cache, if available.

```
SELECT HTML_PLAN ( );
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *GRAPHICAL_PLAN Function [String]* on page 227

HTTP_DECODE function [Web service]

Decodes HTTP encoded strings. This is also known as URL decoding.

Syntax

```
HTTP_DECODE( string )
```

Parameters

- ***string* –** Arbitrary string taken from a URL or URL encoded request body.

Returns

LONG VARCHAR or LONG NVARCHAR

Remarks

This function returns the string argument after replacing all character sequences of the form %*nn*, where *nn* is a hexadecimal value, with the character with code *nn*. In addition, all plus signs (+) are replaced with spaces.

Standards and compatibility

- **SQL/2008 –** Vendor extension.

Examples

```
SELECT HTTP_DECODE('http%3A%2F%2Fdcx.sybase.com')
```

HTTP_ENCODE function [Web service]

Encodes strings for use with HTTP. This is also known as URL encoding.

Syntax

```
HTTP_ENCODE( string )
```

Parameters

- ***string* –** Arbitrary string to be encoded for HTTP transport.

Returns

LONG VARCHAR or LONG NVARCHAR

Remarks

This function returns the string argument after making the following set of substitutions. In addition, all characters with hexadecimal codes less than 20 or greater than 7E are replaced with %*nn*, where *nn* is the character code.

Character	Substitution
space	%20
"	%22
#	%23
%	%25
&	%26
,	%2C
;	%3B
<	%3C
>	%3E
[	%5B
\	%5C
]	%5D
`	%60
{	%7B
|	%7C

Character	Substitution
}	%7D
character codes *nn* that are less than 0x20 and greater than 0x7f	%*nn*

This function supports NCHAR inputs and/or outputs.

Standards and compatibility

- **SQL/2008 –** Vendor extension.

Examples

```
SELECT HTTP_ENCODE('/opt&id=123&text=''oid:c\x09d ef''')
```

HTTP_HEADER function [Web service]

Returns the value of an HTTP request header.

Syntax

```
HTTP_HEADER( header-field-name )
```

Parameters

- ***header-field-name* –** The name of an HTTP request header field.

Returns

LONG VARCHAR.

Note: The result data type is a LONG VARCHAR. If you use HTTP_HEADER in a SELECT INTO statement, you must have an Unstructured Data Analytics Option license or use CAST and set HTTP_HEADER to the correct data type and size.

Remarks

This function returns the value of the named HTTP request header field, or NULL if it does not exist or if it is not called from an HTTP service. It is used when processing an HTTP request via a web service.

Some headers that may be of interest when processing an HTTP web service request include the following:

- **Cookie –** The cookie value(s), if any, stored by the client, that are associated with the requested URI.
- **Referer –** The URL of the page (for example, `http://documents.sample.com:80/index.html`) that contained the link to the requested URI.

- **Host –** The Internet host name or IP address and port number of the resource being requested, as obtained from the original URI given by the user or referring resource (for example, `webserver.sample.com:8082`).
- **User-Agent –** The name of the client application (for example, `Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0`).
- **Accept-Encoding –** A list of encodings for the response that are acceptable to the client application (for example, `gzip, deflate`).

More information about these headers is available at *http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html*.

The following special headers allow access to the elements within the request line of a client request.

- **@HttpMethod –** Returns the type of request being processed. Possible values include DELETE, HEAD, GET, PUT, or POST.
- **@HttpURI –** The full URI of the request, as it was specified in the HTTP request (for example, `/myservice?&id=-123&version=109&lang=en`).
- **@HttpVersion –** The HTTP version of the request (for example, `HTTP/1.0`, or `HTTP/1.1`).
- **@HttpQueryString –** Returns the query portion of the requested URI if it exists (for example, `id=-123&version=109&lang=en`).

Standards and compatibility

- **SQL/2008 –** Vendor extension.

Example

When used within a stored procedure that is called by an HTTP web service, the following example gets the Cookie header value:

```
SET cookie_value = HTTP_HEADER( 'Cookie' );
```

When used within a stored procedure that is called by an HTTP web service, the following example displays the name and values of the HTTP request headers in the database server messages window.

```

BEGIN
  declare header_name long varchar;
  declare header_value long varchar;
  set header_name  = NULL;
header_loop:
  LOOP
    SET header_name = NEXT_HTTP_HEADER( header_name );
    IF header_name IS NULL THEN
      LEAVE header_loop
    END IF;
    SET header_value = HTTP_HEADER( header_name );
    MESSAGE 'HEADER: ', header_name, '=',
            header_value TO CONSOLE;
  END LOOP;
END;
```

HTTP_VARIABLE function [Web service]

Returns the value of an HTTP variable.

Syntax

```
HTTP_VARIABLE( var-name [ , instance [ , attribute ] ] )
```

Parameters

- ***var-name* –** The name of an HTTP variable.
- ***instance* –** If more than one variable has the same name, the instance number of the field instance, or NULL to get the first one. Useful for SELECT lists that permit multiple selections.
- ***attribute* –** In a multi-part request, the attribute can specify a header field name which returns the value of the header for the multi-part name.

  When an attribute is not specified, the returned value is %-decoded and character-set translated to the database character set. UTF %-encoded data is supported in this mode.

  The attribute can also be one of the following modes:

  - **'@BINARY' –** Returns a x-www-form-urlencoded binary data value. This mode indicates that the returned value is %-decoded and not character-set translated. UTF-8 %-encoding is not supported in this mode since %-encoded data are simply decoded into their equivalent byte representation.
  - **'@TRANSPORT' –** Returns the raw HTTP transport form of the value, where %-encodings are preserved.

Returns

LONG VARCHAR.

Note: The result data type is a LONG VARCHAR. If you use HTTP_VARIABLE in a SELECT INTO statement, you must have an Unstructured Data Analytics Option license or use CAST and set HTTP_VARIABLE to the correct data type and size.

Remarks

This function returns the value of the named HTTP variable. It is used when processing an HTTP request within a web service.

If *var-name* does not exist, the return value is NULL.

When the web service request is a POST, and the variable data is posted as multipart/form-data, the HTTP server receives HTTP headers for each individual variable. When the *attribute* parameter is specified, the HTTP_VARIABLE function returns the associated multipart/

form-data header value from the POST request for the particular variable. For a variable representing a file, an attribute of Content-Disposition, Content-Type, and @BINARY would return the filename, media-type, and file contents respectively.

Normally, all input data goes through character set translation between the client (for example, a browser) character set, and the character set of the database. However, if @BINARY is specified for *attribute*, the variable value is returned without going through character set translation or %-decoding. This may be useful when receiving binary data, such as image data, from a client.

This function returns NULL when the specified instance does not exist or when the function is called from outside of an execution of a web service.

Standards and compatibility

- **SQL/2008 –** Vendor extension.

Examples

When used within a stored procedure that is called by an HTTP web service, the following example retrieves the values of the HTTP variables indicated in the sample URL.

```
-- http://sample.com/demo/ShowDetail?product_id=300&customer_id=101
BEGIN
  DECLARE v_customer_id LONG VARCHAR;
  DECLARE v_product_id LONG VARCHAR;
  SET v_customer_id = HTTP_VARIABLE( 'customer_id' );
  SET v_product_id = HTTP_VARIABLE( 'product_id' );
  CALL ShowSalesOrderDetail( v_customer_id, v_product_id );
END;
```

When used within a stored procedure that is called by an HTTP web service, the following statements request the Content-Disposition and Content-Type headers of the image variable:

```
SET v_name = HTTP_VARIABLE( 'image', NULL, 'Content-Disposition' );
SET v_type = HTTP_VARIABLE( 'image', NULL, 'Content-Type' );
```

When used within a stored procedure that is called by an HTTP web service, the following statement requests the value of the image variable in its current character set, that is, without going through character set translation:

```
SET v_image = HTTP_VARIABLE( 'image', NULL, '@BINARY' );
```

IFNULL Function [Miscellaneous]

Returns the first nonnull expression, or NULL.

If the first expression is the NULL value, then the value of the second expression is returned. If the first expression is not NULL, the value of the third expression is returned. If the first expression is not NULL and there is no third expression, then the NULL value is returned.

Syntax

```
IFNULL ( expression1, expression2 [ , expression3 ] )
```

Parameters

Table 44. Parameters

Parameter	Description
expression1	The expression to be evaluated. Its value determines whether *expression2* or *expression3* is returned.
expression2	The return value if *expression1* is NULL
expression3	The return value if *expression1* is not NULL.

Returns

The data type returned depends on the data type of *expression-2* and *expression-3*.

Examples

The following statement returns the value -66:

```
SELECT IFNULL( NULL, -66 ) FROM iq_dummy
```

The following statement returns NULL, because the first expression is not NULL and there is no third expression:

```
SELECT IFNULL( -66, -66 ) FROM iq_dummy
```

Standards and compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

INDEX_COL Function [System]

Returns the name of the indexed column.

Syntax

```
INDEX_COL ( table-name, index-id, key_# [ , user-id ] )
```

Parameters

Parameter	Definition
table-name	A table name.
index-id	The index ID of an index of *table-name*.

Parameter	Definition
key_#	A key in the index specified by *index-id*. This parameter specifies the column number in the index. For a single column index, *key_#* is equal to 0. For a multicolumn index, *key_#* is equal to 0 for the first column, 1 for the second column, and so on.
user-id	The user ID of the owner of *table-name*. If *user-id* is not specified, this value defaults to the caller's user ID..

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ.

See also

- *OBJECT_ID Function [System]* on page 290

INSERTSTR Function [String]

Inserts a string into another string at a specified position.

Syntax

```
INSERTSTR ( numeric-expression, string-expression1, string-
expression2 )
```

Parameters

Parameter	Definition
numeric-expression	The position after which *string-expression2* is to be inserted. Use zero to insert a string at the beginning.
string-expression1	The string into which *string-expression2* is to be inserted.
string-expression2	The string to be inserted.

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **INSERTSTR** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **INSERTSTR** to the correct data type and size.

Example

The following statement returns the value “backoffice”:

```
SELECT INSERTSTR( 0, 'office ', 'back' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported in Adaptive Server Enterprise. The STUFF function is equivalent and is supported in both Adaptive Server Enterprise and SAP Sybase IQ.

INTTOHEX Function [Data Type Conversion]

Returns the hexadecimal equivalent of a decimal integer.

Syntax

```
INTTOHEX ( integer-expression )
```

Parameters

Parameter	Description
integer-expression	The integer to be converted to hexadecimal.

Returns

VARCHAR

Examples

The following statement returns the value 3B9ACA00:

```
SELECT INTTOHEX( 1000000000 ) FROM iq_dummy
```

The following statement returns the value 00000002540BE400:

```
SELECT INTTOHEX ( 10000000000) FROM iq_dummy
```

Usage

If data conversion of input to **INTTOHEX** conversion fails, SAP Sybase IQ returns an error, unless the `CONVERSION_ERROR` option is OFF. In that case, the result is NULL.

The database option `ASE_FUNCTION_BEHAVIOR` specifies that output of SAP Sybase IQ functions, including **INTTOHEX** and **HEXTOINT**, be consistent with the output of Adaptive Server Enterprise functions. The default value of `ASE_FUNCTION_BEHAVIOR` is OFF.

When the `ASE_FUNCTION_BEHAVIOR` option is disabled (the value is OFF):

- The output of **INTTOHEX** is compatible with SQL Anywhere.

- Depending on the input, the output of **INTTOHEX** can be 8 digits or 16 digits and is left padded with zeros; the return data type is `VARCHAR`.
- The output of **INTTOHEX** does not have a '0x' or '0X' prefix.
- The input to **INTTOHEX** can be up to a 64-bit integer.

When the `ASE_FUNCTION_BEHAVIOR` option is enabled (the value is ON):

- The output of **INTTOHEX** is compatible with Adaptive Server Enterprise.
- The output of **INTTOHEX** is always 8 digits and is left-padded with zeros; the return data type is `VARCHAR`.
- The output of **INTTOHEX** does not have a '0x' or '0X' prefix.
- SAP Sybase IQ **INTTOHEX** assumes input is a 32-bit signed integer; a larger value can overflow and a conversion error can result. For example, the statement:

  ```
  SELECT INTTOHEX( 1000000000 ) FROM iq_dummy
  ```

  returns the value `3B9ACA00`. But the statement:

  ```
  SELECT INTTOHEX( 10000000000 ) FROM iq_dummy
  ```

  results in a conversion error.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *BIGINTTOHEX Function [Data Type Conversion]* on page 164
- *HEXTOBIGINT Function [Data Type Conversion]* on page 231
- *HEXTOINT Function [Data Type Conversion]* on page 233

ISDATE Function [Date and Time]

Tests whether a string argument can be converted to a date.

If a conversion is possible, the function returns 1; otherwise, it returns 0. If the argument is null, 0 is returned.

Syntax

```
ISDATE ( string )
```

Parameters

Table 45. Parameters

Parameter	Description
string	The string to be analyzed to determine whether the string represents a valid date.

Returns

INT

Example

The following example tests whether the `birth_date` column holds valid dates, returning invalid dates as NULL, and valid dates in `date` format.

```
select birth_date from MyData;
------------------------------
1990/32/89
0101/32/89
1990/12/09
```

```
select
  case when isdate(birth_date)=0 then NULL
  else cast(birth_date as date)
  end
  from MyData;
------------------------------------
(NULL)
(NULL)
1990-12-09
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase:
  - SQL Anywhere uses ISO 8601 date interchange formats.
  - Not supported by Adaptive Server Enterprise.

ISNULL Function [Miscellaneous]

Returns the value of the first non-NULL expression in the parameter list.

At least two expressions must be passed to the function.

Syntax

```
ISNULL ( expression, expression [ …, expression ] )
```

Parameters

Table 46. Parameters

Parameter	Description
expression	An expression to be tested against NULL.

Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the result in the type used for the comparison. If the database server cannot find a common comparison type, an error is returned.

Example

The following statement returns the value -66:

```
SELECT ISNULL( NULL ,-66, 55, 45, NULL, 16 ) FROM iq_dummy
```

Usage

The **ISNULL** function is the same as the **COALESCE** function.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *COALESCE Function [Miscellaneous]* on page 174

ISNUMERIC Function [Miscellaneous]

Tests whether a string argument can be converted to a numeric.

If a conversion is possible, the function returns 1; otherwise, it returns 0. If the argument is null, 0 is returned.

Syntax

ISNUMERIC (*string*)

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
string	The string to be analyzed to determine whether the string represents a valid numeric value

Returns
INT

Usage
For optimal performance, avoid using **ISNUMERIC** in predicates, where it is processed by the SQL Anywhere portion of the product and cannot take advantage of the performance features of SAP Sybase IQ.

Example
The following example tests whether the `height_in_cms` column holds valid numeric data, returning invalid numeric data as NULL, and valid numeric data in `int` format.

```
data height_in_cms
------------------------
asde
asde
180
156
```

```
select case
   when isnumeric(height_in_cms)=0
   then NULL
   else cast(height_in_cms as int)
   end
from MyData
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

LAG Function [Analytical]

An inter-row function that returns the value of an attribute in a previous row in the table or table partition.

Syntax

LAG (*value_expr*) [, *offset* [, *default*]]) **OVER** ([**PARTITION BY** *window partition*] **ORDER BY** *window ordering*)

Parameters

Parameter	Description
value_expr	Table column or expression defining the offset data to return from the table.
offset	The number of rows above the current row, expressed as a non-negative exact numeric literal, or as a SQL variable with exact numeric data. The permitted range is 0 to 231.
default	The value to return if the *offset* value goes beyond the scope of the cardinality of the table or partition.
window partition	(Optional) One or more value expressions separated by commas indicating how you want to divide the set of result rows.
window ordering	Defines the expressions for sorting rows within window partitions, if specified, or within the result set if you did not specify a window partition.

Usage

The **LAG** function requires an **OVER** (**ORDER_BY**) window specification. The window partitioning clause in the **OVER** (**ORDER_BY**) clause is optional. The **OVER** (**ORDER_BY**) clause must not contain a window frame **ROWS**/**RANGE** specification.

You cannot define an analytic expression in *value_expr*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *value_expr*.

You must enter a non-negative numeric data type for *offset*. Entering **0** returns the current row. Entering a negative number generates an error.

The default value of *default* is **NULL**. The data type of *default* must be implicitly convertible to the data type of the *value_expr* value or else SAP Sybase IQ generates a conversion error.

Example

The following example returns salary data from the Employees table, partitions the data by department ID, and orders the data according to employee start date. The **LAG** function returns the salary from the previous row (a physical offset of one row) and displays it under the **LAG (Salary)** column:

```
SELECT DepartmentID dID, StartDate, Salary, LAG(Salary, 1)
OVER(PARTITION BY dID ORDER BY StartDate) FROM Employees ORDER BY
1,2;
```

The returned result set is:

```
dID        StartDate    Salary      Lag(Salary)
=========  ===========  ==========  =============
100        1984-08-28   45,700.000  NULL
100        1985-01-01   62,000.000  45,700.000
100        1985-06-17   57,490.000  62,000.000
100        1986-06-07   72,995.000  57,490.000
100        1986-07-01   48,023.690  72,995.000
...
200        1985-02-03   38,500.000  NULL
200        1985-12-06   54,800.000  38,500.000
200        1987-02-19   39,300.000  54,800.000
200        1987-07-10   49,500.000  39,300.000
200        1988-10-04   54,600.000  49,500.000
200        1988-11-12   39,800.000  54,600.000
...
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

See also

- *LEAD Function [Analytical]* on page 257

LAST_VALUE Function [Aggregate]

Returns the last value from a set of values.

Syntax

LAST_VALUE (*expression* [IGNORE NULLS | RESPECT NULLS])

OVER (*window-spec*)

Parameters

Parameter	Definition
expression	The expression on which to determine the last value in an ordered set

Returns

Data type of the argument.

Usage

LAST_VALUE returns the last value in a set of values, which is usually an ordered set. If the last value in the set is null, then the function returns NULL unless you specify IGNORE NULLS. If you specify IGNORE NULLS, then **LAST_VALUE** returns the last non-null value in the set, or NULL if all values are null.

The data type of the returned value is the same as that of the input value.

You cannot use **LAST_VALUE** or any other analytic function for expression. That is, you cannot nest analytic functions, but you can use other built-in function expressions for expression.

If the window-spec does not contain an **ORDER BY** expression, or if the **ORDER BY** expression is not precise enough to guarantee a unique ordering, then the result is arbitrary. If there is no window-spec, then the result is arbitrary.

You can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Note: DISTINCT is not supported.

Example

The following example returns the salary of each employee, plus the name of the employee with the highest salary in their department:

```
SELECT GivenName + ' ' + Surname AS employee_name,
    Salary, DepartmentID,
    LAST_VALUE( employee_name ) OVER Salary_Window AS
highest_paid
FROM Employees
WINDOW Salary_Window AS ( PARTITION BY DepartmentID ORDER BY Salary
    RANGE BETWEEN UNBOUNDED PRECEDING
    AND UNBOUNDED FOLLOWING )
ORDER BY DepartmentID DESC;
```

The returned result set is:

Table 47. LAST_VALUE result set

employee_name	Salary	DepartmentID	highest_paid
Michael Lynch	24,903.000	500	Jose Martinez
Joseph Barker	27,290.000	500	Jose Martinez
Sheila Romero	27,500.000	500	Jose Martinez
Felicia Kuo	28,200.000	500	Jose Martinez
Jeannette Bertrand	29,800.000	500	Jose Martinez
Jane Braun	34,300.000	500	Jose Martinez
Anthony Rebeiro	34,576.000	500	Jose Martinez
Charles Crowley	41,700.000	500	Jose Martinez
Jose Martinez	55,500.800	500	Jose Martinez
Doug Charlton	28,300.000	400	Scott Evans

employee_name	Salary	DepartmentID	highest_paid
Elizabeth Lambert	29,384.000	400	Scott Evans
Joyce Butterfield	34,011.000	400	Scott Evans
Robert Nielsen	34,889.000	400	Scott Evans
Alex Ahmed	34,992.000	400	Scott Evans
Ruth Wetherby	35,745.000	400	Scott Evans
...	...	...	...

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

See also

- *Windowing Aggregate Function Usage* on page 83

LCASE Function [String]

Converts all characters in a string to lowercase.

Syntax

```
LCASE ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string to be converted to lowercase.

Returns

CHAR

NCHAR

LONG VARCHAR

VARCHAR

NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **LCASE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **LCASE** to the correct data type and size.

Example

The following statement returns the value “lower case”:

```
SELECT LCASE( 'LOWER CasE' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—**LCASE** is not supported in Adaptive Server Enterprise; you can use **LOWER** to get the same functionality.

See also

- *LEFT Function [String]* on page 259
- *LOWER Function [String]* on page 269
- *REPLACE Function [String]* on page 321
- *REVERSE Function [String]* on page 324
- *RIGHT Function [String]* on page 325
- *UCASE Function [String]* on page 366
- *UPPER Function [String]* on page 367

LEAD Function [Analytical]

An inter-row function that returns the value of an attribute in a subsequent row in the table or table partition.

Syntax

```
LEAD (value_expr) [, offset [, default]]) OVER ([PARTITION BY window
partition] ORDER BY window ordering)
```

Parameters

Parameter	Description
value_expr	Table column or expression defining the offset data to return from the table.
offset	The number of rows below the current row, expressed as a non-negative exact numeric literal, or as a SQL variable with exact numeric data. The permitted range is 0 to 231.
default	The value to return if the *offset* value goes beyond the scope of the table or partition.

Parameter	Description
window partition	(Optional) One or more value expressions separated by commas indicating how you want to divide the set of result rows.
window ordering	Defines the expressions for sorting rows within window partitions, if specified, or within the result set if you did not specify a window partition.

Usage

The **LEAD** function requires an **OVER** (**ORDER_BY**) window specification. The window partitioning clause in the **OVER** (**ORDER_BY**) clause is optional. The **OVER** (**ORDER_BY**) clause must not contain a window frame **ROWS**/**RANGE** specification.

You cannot define an analytic expression in *value_expr*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *value_expr*.

You must enter a non-negative numeric data type for *offset*. Entering **0** returns the current row. Entering a negative number generates an error.

The default value of *default* is **NULL**. The data type of *default* must be implicitly convertible to the data type of the *value_expr* value or else SAP Sybase IQ generates a conversion error.

Example

The following example returns salary data from the Employees table, partitions the data by department ID, and orders the data according to employee start date. The **LEAD** function returns the salary from the next row (a physical offset of one row) and displays it under the **LEAD (Salary)** column:

```
SELECT DepartmentID dID, StartDate, Salary, LEAD(Salary, 1)
OVER(PARTITION BY dID ORDER BY StartDate) FROM  Employees ORDER BY
1,2;
```

The returned result set is:

```
dID        StartDate    Salary      Lead(Salary)
=========  ===========  ==========  =============
100        1984-08-28    45,700.000    62,000.000
100        1985-01-01    62,000.000    57,490.000
100        1985-06-17    57,490.000    72,995.000
100        1986-06-07    72,995.000    48,023.690
...
100        1990-08-19    54,900.000    NULL
200        1985-02-03    38,500.000    39,300.000
200        1987-02-19    39,300.000    49,500.000
200        1987-07-10    49,500.000    54,600.000
200        1988-11-28    46,200.000    34,892.000
200        1989-06-01    34,892.000    87,500.000
...
200        1993-08-12    47,653.000    NULL
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

See also

- *LAG Function [Analytical]* on page 252

LEFT Function [String]

Returns a specified number of characters from the beginning of a string.

Syntax

```
LEFT ( string-expression, numeric-expression )
```

Parameters

Table 48. Parameters

Parameter	Description
string-expression	The string.
numeric-expression	The number of characters to return.

Returns

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **LEFT** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **LEFT** to the correct data type and size.

Example

The following statement returns the value “choco”:

```
SELECT LEFT( 'chocolate', 5 ) FROM iq_dummy
```

Usage

If the string contains multibyte characters, and the proper collation is being used, the number of bytes returned may be greater than the specified number of characters.

Note: The result data type of a **LEFT** function is a `LONG VARCHAR`. If you use **LEFT** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics option license or use **CAST** and set **LEFT** to the correct data type and size.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *LCASE Function [String]* on page 256
- *LOWER Function [String]* on page 269
- *REPLACE Function [String]* on page 321
- *REVERSE Function [String]* on page 324
- *RIGHT Function [String]* on page 325
- *UCASE Function [String]* on page 366
- *UPPER Function [String]* on page 367

LEN Function [String]

Takes one argument as an input of type `BINARY` or `STRING` and returns the number of characters, as defined by the database's collation sequence, of a specified string expression, excluding trailing blanks.

The result may differ from the string’s byte length for multi-byte character sets.

`BINARY` and `VARBINARY` are also allowed, in which case LEN() returns the number of bytes of the input.

LEN is an alias of **LENGTH** function

Syntax

```
LEN ( string_expr )
```

Parameters

Table 49. Parameters

Parameters	Description
string_expr	The string expression to be evaluated.

Example

The following example returns the value 3152:

```
select len(Photo) from Productswhere ID = 500
```

Usage

This function is the equivalent of **CHAR_LENGTH** (*string_expression*).

Permissions
Any user can execute **LEN**.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.

See also

- *BIT_LENGTH Function [String]* on page 165
- *BYTE_LENGTH Function [String]* on page 166
- *CHAR_LENGTH Function [String]* on page 171
- *COL_LENGTH Function [System]* on page 175
- *DATALENGTH Function [System]* on page 189
- *LENGTH Function [String]* on page 261
- *OBJECT_NAME Function [System]* on page 291
- *OCTET_LENGTH Function [String]* on page 292
- *STR_REPLACE Function [String]* on page 352

LENGTH Function [String]

Returns the number of characters in the specified string.

Syntax

```
LENGTH ( string-expression )
```

Parameters

Table 50. Parameters

Parameter	Description
string-expression	The string.

Returns
INT

Example
The following statement returns the value 9:

```
SELECT LENGTH( 'chocolate' ) FROM iq_dummy
```

Usage
If the string contains multibyte characters, and the proper collation is being used, **LENGTH** returns the number of characters, not the number of bytes. If the string is of `BINARY` data type, the **LENGTH** function behaves as **BYTE_LENGTH**.

The **LENGTH** function is the same as the **CHAR_LENGTH** function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise. Use the **CHAR_LENGTH** function instead.

See also

- *BIT_LENGTH Function [String]* on page 165
- *BYTE_LENGTH Function [String]* on page 166
- *CHAR_LENGTH Function [String]* on page 171
- *COL_LENGTH Function [System]* on page 175
- *DATALENGTH Function [System]* on page 189
- *LEN Function [String]* on page 260
- *OBJECT_NAME Function [System]* on page 291
- *OCTET_LENGTH Function [String]* on page 292
- *STR_REPLACE Function [String]* on page 352

LIST function [Aggregate]

Returns a delimited list of values for every row in a group.

Syntax

```
LIST(
[ALL | DISTINCT] string-expresssion
[, 'delimiter-string']
[ORDER BY order-by-expression [ ASC | DESC ], ... ] )
```

Note: CIS functional compensation performance considerations apply.

Parameters

- **string-expression –** A string expression, usually a column name. When ALL is specified (the default), for each row in the group, the value of string-expression is added to the result string, with values separated by delimiter-string. When DISTINCT is specified, only unique string-expression values are added.
- **delimiter-string –** A delimiter string for the list items. The default setting is a comma. There is no delimiter if a value of NULL or an empty string is supplied. The delimiter-string must be a constant.
- **order-by-expression –** Order the items returned by the function. There is no comma preceding this argument, which makes it easy to use in the case where no delimiter-string is supplied.

  order-by-expression cannot be an integer literal. However, it can be a variable that contains an integer literal.

When an ORDER BY clause contains constants, they are interpreted by the optimizer and then replaced by an equivalent ORDER BY clause. For example, the optimizer interprets ORDER BY 'a' as ORDER BY expression.

A query block containing more than one aggregate function with valid ORDER BY clauses can be executed if the ORDER BY clauses can be logically combined into a single ORDER BY clause. For example, the following clauses:

```
ORDER BY expression1, 'a', expression2
```

```

ORDER BY expression1, 'b', expression2, 'c', expression3
```

are subsumed by the clause:

```
ORDER BY expression1, expression2, expression3
```

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **LIST** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **LIST** to the correct data type and size.

Remarks

The LIST function returns the concatenation (with delimiters) of all the non-NULL values of X for each row in the group. If there does not exist at least one row in the group with a definite X-value, then LIST(X) returns the empty string.

NULL values and empty strings are ignored by the LIST function.

A LIST function cannot be used as a window function, but it can be used as input to a window function.

This function supports NCHAR inputs and/or outputs.

Standards and Compatibility

- **SQL/2008 –** Vendor extension.

  SAP Sybase IQ supports SQL/2008 language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions that are not column references.

  SAP Sybase IQ does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SAP Sybase IQdoes not permit the arguments of an aggregate function to include both a column reference from the query block containing the LIST function, combined with an outer reference.

Examples

This statement returns the value 487 Kennedy Court, 547 School Street.

```
SELECT LIST( Street ) FROM Employees
WHERE GivenName = 'Thomas';
```

This statement lists employee IDs. Each row in the result set contains a comma-delimited list of employee IDs for a single department.

```
SELECT LIST( EmployeeID )
FROM Employees
GROUP BY DepartmentID;
```

LIST(EmployeeID)
102,105,160,243,247,249,266,278,...
129,195,299,467,641,667,690,856,...
148,390,586,757,879,1293,1336,...
184,207,318,409,591,888,992,1062,...
191,703,750,868,921,1013,1570,...

This statement sorts the employee IDs by the last name of the employee:

```
SELECT LIST( EmployeeID ORDER BY Surname ) AS "Sorted IDs"
FROM Employees
GROUP BY DepartmentID;
```

Sorted IDs
1013,191,750,921,868,1658,...
1751,591,1062,1191,992,888,318,...
1336,879,586,390,757,148,1483,...
1039,129,1142,195,667,1162,902,...
160,105,1250,247,266,249,445,...

This statement returns semicolon-separated lists. Note the position of the ORDER BY clause and the list separator:

```
SELECT LIST( EmployeeID, ';' ORDER BY Surname ) AS "Sorted IDs"
FROM Employees
GROUP BY DepartmentID;
```

Sorted IDs
1013;191;750;921;868;1658;703;...
1751;591;1062;1191;992;888;318;...
1336;879;586;390;757;148;1483;...

Sorted IDs
1039;129;1142;195;667;1162;902; ...
160;105;1250;247;266;249;445;...

Be sure to distinguish the previous statement from the following statement, which returns comma-separated lists of employee IDs sorted by a compound sort-key of (Surname, ';'):

```
SELECT LIST( EmployeeID ORDER BY Surname, ';' ) AS "Sorted IDs"
FROM Employees
GROUP BY DepartmentID;
```

LN Function [Numeric]

Returns the natural logarithm of the specified expression.

Syntax

```
LN ( numeric-expression )
```

Parameters

Parameter	Description
expression	Is a column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, the **LN** function generates an error. The return value is of `DOUBLE` data type.

Usage

LN takes one argument. For example, **LN** (*20*) returns 2.995732.

The **LN** function is an alias of the **LOG** function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise. Use the LOG function instead.

See also

- *LOG Function [Numeric]* on page 267
- *LOG10 Function [Numeric]* on page 268

LOCATE Function [String]

Returns the position of one string within another.

Syntax

```
LOCATE ( string-expression1, string-expression2
[ , numeric-expression ] )
```

Parameters

Parameter	Description
string-expression1	The string to be searched.
string-expression2	The string for which you are searching. This string is limited to 255 bytes.
numeric-expression	The character position at which to begin the search in the string. The first character is position 1. If the starting offset is negative, **LOCATE** returns the last matching string offset, rather than the first. A negative offset indicates how much of the end of the string to exclude from the search. The number of bytes excluded is calculated as (-1 * offset) - 1.

The *numeric-expression* is a 32 bit signed integer for `CHAR`, `VARCHAR`, and `BINARY` columns.

Returns

INT

Examples

The following statement returns the value 8:

```
SELECT LOCATE( 'office party this week – rsvp as soon as possible',
'party', 2 ) FROM iq_dummy
```

In the second example, the *numeric-expression* starting offset for the search is a negative number.

```
CREATE TABLE t1(name VARCHAR(20), dirname VARCHAR(60));
  INSERT INTO t1     VALUES(‘m1000’,’c:\test\functions\locate.sql’);
  INSERT INTO t1     VALUES(‘m1001’,’d:\test\functions\trim.sql’);
COMMIT;

SELECT LOCATE(dirname, ‘\’, -1), dirname FROM t1;
```

The result is:

```
18   c:\test\functions\locate.sql
18   d:\test\functions\trim.sql
```

Usage

If *numeric-expression* is specified, the search starts at that offset into the string being searched.

If *numeric-expression* is not specified, **LOCATE** returns only the position of the first instance of the specified string.

The first string can be a long string (longer than 255 bytes), but the second is limited to 255 bytes. If a long string is given as the second argument, the function returns a NULL value.

If any of the arguments is NULL, the result is NULL.

Searching for a zero-length string returns 1.

If the string does not contain the specified string, the **LOCATE** function returns zero (0).

All the positions or offsets, returned or specified, in the **LOCATE** function are always character offsets and may be different from the byte offset for multibyte data.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *LOCATE Function* in *Unstructured Data Analytics*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *PATINDEX Function [String]* on page 292
- *LIKE Conditions* on page 47

LOG Function [Numeric]

Returns the natural logarithm of a number.

LN is an alias of **LOG**.

Syntax

```
LOG ( numeric-expression )
```

Parameters

Table 51. Parameters

Parameter	Description
numeric-expression	The number.

Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Example

The following statement returns the value 3.912023:

```
SELECT LOG( 50 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *LN Function [Numeric]* on page 265
- *LOG10 Function [Numeric]* on page 268

LOG10 Function [Numeric]

Returns the base 10 logarithm of a number.

Syntax

```
LOG10 ( numeric-expression )
```

Parameters

Table 52. Parameters

Parameter	Description
numeric-expression	The number.

Returns

This function converts its argument to DOUBLE, and performs the computation in double-precision floating point. If the parameter is NULL, the result is NULL.

Example

The following statement returns the value 1.698970.

```
SELECT LOG10( 50 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *LN Function [Numeric]* on page 265
- *LOG Function [Numeric]* on page 267

LOWER Function [String]

Converts all characters in a string to lowercase.

Syntax

```
LOWER ( string-expression )
```

Parameters

Table 53. Parameters

Parameter	Description
string-expression	The string to be converted.

Returns

CHAR

NCHAR

LONG VARCHAR

VARCHAR

NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **LOWER** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **LOWER** to the correct data type and size.

Example

The following statement returns the value “lower case”:

```
SELECT LOWER( 'LOWER CasE' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *LCASE Function [String]* on page 256
- *LEFT Function [String]* on page 259
- *REPLACE Function [String]* on page 321
- *REVERSE Function [String]* on page 324
- *RIGHT Function [String]* on page 325
- *UCASE Function [String]* on page 366
- *UPPER Function [String]* on page 367

LTRIM Function [String]

Removes leading blanks from a string.

Syntax

```
LTRIM ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string to be trimmed.

Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **LTRIM** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **LTRIM** to the correct data type and size.

Example

The following statement returns the value “Test Message” with all leading blanks removed:

```
SELECT LTRIM( '     Test Message' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *RTRIM Function [String]* on page 331

MAX Function [Aggregate]

Returns the maximum *expression* value found in each group of rows.

Syntax

```
MAX ( expression
| DISTINCT column-name )
```

Parameters

Parameter	Description
expression	The expression for which the maximum value is to be calculated. This is commonly a column name.
DISTINCT column-name	Returns the same as MAX (expression), and is included for completeness.

Returns

The same data type as the argument.

Example

The following statement returns the value 138948.000, representing the maximum salary in the `Employees` table:

```
SELECT MAX ( Salary )
FROM Employees
```

Usage

Rows where *expression* is NULL are ignored. Returns NULL for a group containing no rows.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *Windowing Aggregate Function Usage* on page 83
- *MIN Function [Aggregate]* on page 273

MEDIAN Function [Aggregate]

Returns the median of an expression.

Syntax 1

```
MEDIAN([ALL | DISTINCT] expression)
```

Syntax 2

```
MEDIAN([ALL | DISTINCT] expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
expression	A numeric expression for which a median value is to be computed.

Usage

The median is the number separating the higher half of a sample, a population, or a probability distribution, from the lower half.

The data type of the returned value is the same as that of the input value. NULLs are ignored in the calculation of the median value. You can use the optional keyword **DISTINCT** to eliminate duplicate values before the aggregate function is applied. **ALL**, which performs the operation on all rows, is the default.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Note: The *window-spec* cannot contain a **ROW**, **RANGE** or **ORDER BY** specification; *window-spec* can only specify a **PARTITION** clause. DISTINCT is not supported if a **WINDOW** clause is used.

Example

The following query returns the median salary for each department in Florida:

```
SELECT DepartmentID, Surname, Salary,
MEDIAN(Salary) OVER (PARTITION BY DepartmentID) "Median"
FROM Employees
WHERE State IN ('FL')
```

The returned result is:

Table 54. MEDIAN result set

DepartmentID	Surname	Salary	Median
100	Lull	87,900.000	73,870.000
100	Gowda	59,840.000	73,870.000
200	Sterling	64,900.000	76,200.000
200	Kelly	87,500.000	76,200.000
300	Litton	58,930.000	58,930.000
400	Francis	53,870.000	38,70.000
400	Charlton	28,300.000	53,870.000
400	Evans	68,940.000	53,870.000

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

See also

- *Windowing Aggregate Function Usage* on page 83

MIN Function [Aggregate]

Returns the minimum expression value found in each group of rows.

Syntax

```
MIN ( expression
| DISTINCT column-name )
```

Parameters

Parameter	Description
expression	The expression for which the minimum value is to be calculated. This is commonly a column name.
DISTINCT column-name	Returns the same as MIN (expression), and is included for completeness.

Returns

The same data type as the argument.

Example

The following statement returns the value 24903.000, representing the minimum salary in the Employees table:

```
SELECT MIN ( Salary )
FROM Employees
```

Usage

Rows where *expression* is NULL are ignored. Returns NULL for a group containing no rows.

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *Windowing Aggregate Function Usage* on page 83
- *MAX Function [Aggregate]* on page 271

MINUTE Function [Date and Time]

Returns a number from 0 to 59 corresponding to the minute component of the specified date/time value.

Syntax

```
MINUTE ( datetime-expression )
```

Parameters

Parameter	Description
datetime-expression	The date/time value.

Returns

SMALLINT

Example

The following statement returns the value 22:

```
SELECT MINUTE( '1998-07-13 12:22:34' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

MINUTES Function [Date and Time]

Returns the number of minutes since an arbitrary date and time, the number of whole minutes between two specified times, or adds the specified integer-expression number of minutes to a time.

Syntax

```
MINUTES ( datetime-expression
| datetime-expression, datetime-expression
| datetime-expression, integer-expression )
```

Parameters

Parameter	Description
datetime-expression	A date and time.
integer-expression	The number of minutes to be added to the *datetime-expression.* If *integer-expression* is negative, the appropriate number of minutes are subtracted from the date/time. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a `datetime` data type

Returns

INT

TIMESTAMP

Examples

Returns the value 1051125487:

```
SELECT MINUTES( '1998-07-13 06:07:12' ) FROM iq_dummy
```

Returns the value 240, to signify the difference between the two times:

```
SELECT MINUTES( '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' ) FROM iq_dummy
```

Returns the datetime value 1999-05-12 21:10:07.000:

```
SELECT MINUTES( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5) FROM iq_dummy
```

Usage

The second syntax returns the number of whole minutes from the first date/time to the second date/time. The number might be negative.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported in Adaptive Server Enterprise.

See also

- *CAST Function [Data Type Conversion]* on page 167
- *CONVERT Function [Data Type Conversion]* on page 178
- *HOURS Function [Date and Time]* on page 235
- *MONTHS Function [Date and Time]* on page 278
- *REPLACE Function [String]* on page 321
- *SECOND Function [Date and Time]* on page 331
- *WEEKS Function [Date and Time]* on page 375
- *YEAR Function [Date and Time]* on page 380
- *YEARS Function [Date and Time]* on page 381

MOD Function [Numeric]

Returns the remainder when one whole number is divided by another.

Syntax

```
MOD ( dividend, divisor )
```

Parameters

Parameters	Description
dividend	The dividend, or numerator of the division.
divisor	The divisor, or denominator of the division.

Returns

SMALLINT

INT

NUMERIC

Example

The following statement returns the value 2:

```
SELECT MOD( 5, 3 ) FROM iq_dummy
```

Usage

Division involving a negative *dividend* gives a negative or zero result. The sign of the *divisor* has no effect.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported in Adaptive Server Enterprise. The % operator is used as a modulo operator in Adaptive Server Enterprise.

See also

- *REMAINDER Function [Numeric]* on page 319

MONTH Function [Date and Time]

Returns a number from 1 to 12 corresponding to the month of the given date.

Syntax

```
MONTH ( date-expression )
```

Parameters

Parameters	Description
date-expression	A date/time value.

Returns
SMALLINT

Example
The following statement returns the value 7:

```
SELECT MONTH( '1998-07-13' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

MONTHNAME Function [Date and Time]

Returns the name of the month from the specified date expression.

Syntax

```
MONTHNAME ( date-expression )
```

Parameters

Parameter	Description
date-expression	The datetime value.

Returns
VARCHAR

Example
The following statement returns the value **September**, when the `DATE_ORDER` option is set to the default value of *ymd*.

```
SELECT MONTHNAME( '1998-09-05' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

MONTHS Function [Date and Time]

Returns the number of months since an arbitrary starting date/time or the number of months between two specified date/times, or adds the specified integer-expression number of months to a date/time.

Syntax

```
MONTHS ( date-expression
| date-expression, datetime-expression
| date-expression, integer-expression )
```

Parameters

Parameter	Description
date-expression	A date and time.
integer-expression	The number of months to be added to the *date-expression*. If *integer-expression* is negative, the appropriate number of months are subtracted from the date/time value. If you supply an integer expression, the *date-expression* must be explicitly cast as a `datetime` data type.

Returns
INT

TIMESTAMP

Examples

The following statement returns the value 23982:

```
SELECT MONTHS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 2, to signify the difference between the two dates:

```
SELECT MONTHS( '1999-07-13 06:07:12',
    '1999-09-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-10-12 21:05:07.000:

```
SELECT MONTHS( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5) FROM iq_dummy
```

Usage

The first syntax returns the number of months since an arbitrary starting date. This number is often useful for determining whether two date/time expressions are in the same month in the same year.

```
MONTHS( invoice_sent ) = MONTHS( payment_received )
```

Comparing the **MONTH** function would incorrectly include a payment made 12 months after the invoice was sent.

The second syntax returns the number of months from the first date to the second date. The number might be negative. It is calculated from the number of the first days of the month between the two dates. Hours, minutes and seconds are ignored.

The third syntax adds *integer-expression* months to the given date. If the new date is past the end of the month (such as **MONTHS** ('1992-01-31', 1)) the result is set to the last day of the month. If *integer-expression* is negative, the appropriate number of months are subtracted from the date. Hours, minutes and seconds are ignored.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *CAST Function [Data Type Conversion]* on page 167
- *CONVERT Function [Data Type Conversion]* on page 178
- *HOURS Function [Date and Time]* on page 235
- *MINUTES Function [Date and Time]* on page 275
- *REPLACE Function [String]* on page 321
- *SECOND Function [Date and Time]* on page 331
- *WEEKS Function [Date and Time]* on page 375
- *YEAR Function [Date and Time]* on page 380

- *YEARS Function [Date and Time]* on page 381

NEWID Function [Miscellaneous]

Generates a UUID (Universally Unique Identifier) value.

The returned UUID value is a binary. A UUID is the same as a GUID (Globally Unique Identifier).

Syntax

```
NEWID ( )
```

Parameters

There are no parameters associated with **NEWID**().

Returns

UNIQUEIDENTIFIER

Example

The following statement creates the table `t1` and then updates the table, setting the value of the column `uid_col` to a unique identifier generated by the **NEWID** function, if the current value of the column is NULL.

```
CREATE TABLE t1 (uid_col int);
UPDATE t1
    SET uid_col = NEWID()
      WHERE uid_col IS NULL
```

If you execute the following statement,

```
SELECT NEWID()
```

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

Usage

The **NEWID**() function generates a unique identifier value.

UUIDs can be used to uniquely identify objects in a database. The values are generated such that a value produced on one computer does not match that produced on another, hence they can also be used as keys in replication and synchronization environments.

The **NEWID** function is supported only in the following positions:

- **SELECT** list of a top level query block
- **SET** clause of an **UPDATE** statement
- **VALUES** clause of **INSERT...VALUES**

You can use a value generated by the **NEWID** function as a column default value in a table.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *Binary Data Types* on page 394
- *STRTOUUID Function [String]* on page 355
- *UUIDTOSTR Function [String]* on page 369
- *Character Data Types* on page 385
- *Binary Data Types* on page 819

NEXT_CONNECTION Function [System]

Returns the next connection number, or the first connection if the parameter is NULL.

Syntax

```
NEXT_CONNECTION ( {connection-id }, {database-id } )
```

Note: CIS functional compensation performance considerations apply.

Returns

INT

Parameters

Parameter	Description
connection-id	An integer, usually returned from a previous call to **NEXT_CONNECTION**. If *connection-id* is NULL, **NEXT_CONNECTION** returns the most recent connection ID.
database-id	An integer representing one of the databases on the current server. If you supply no *database-id*, the current database is used. If you supply NULL, then **NEXT_CONNECTION** returns the next connection regardless of database.

Usage

You can use **NEXT_CONNECTION** to enumerate the connections to a database. To get the first connection, pass NULL; to get each subsequent connection, pass the previous return value. The function returns NULL when there are no more connections.

NEXT_CONNECTION can be used to enumerate the connections to a database. Connection IDs are generally created in monotonically increasing order. This function returns the next connection ID in reverse order.

To get the connection ID value for the most recent connection, enter NULL as the *connection-id*. To get the subsequent connection, enter the previous return value. The function returns NULL when there are no more connections in the order.

NEXT_CONNECTION is useful if you want to disconnect all the connections created before a specific time. However, because **NEXT_CONNECTION** returns the connection IDS in reverse order, connections made after the function is started are not returned. If you want to ensure that all connections are disconnected, prevent new connections from being created before you run **NEXT_CONNECTION**.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

Examples

The following statement returns an identifier for the first connection on the current database. The identifier is an integer value like 10.

```
SELECT NEXT_CONNECTION( NULL );
```

The following statement returns a value like 5.

```
SELECT NEXT_CONNECTION( 10 );
```

The following call returns the next connection ID in reverse order from the specified *connection-id* on the current database.

```
SELECT NEXT_CONNECTION( connection-id );
```

The following call returns the next connection ID in reverse order from the specified *connection-id* (regardless of database).

```
SELECT NEXT_CONNECTION( connection-id, NULL );
```

The following call returns the next connection ID in reverse order from the specified *connection-id* on the specified database.

```
SELECT NEXT_CONNECTION( connection-id, database-id );
```

The following call returns the first (earliest) connection (regardless of database).

```
SELECT NEXT_CONNECTION( NULL, NULL );
```

The following call returns the first (earliest) connection on the specified database.

```
SELECT NEXT_CONNECTION( NULL, database-id );
```

NEXT_DATABASE Function [System]

Returns the next database ID number, or the first database if the parameter is NULL.

Syntax

```
NEXT_DATABASE ( { NULL | database-id } )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
database-id	An integer that specifies the ID number of the database.

Returns

INT

Examples

The following statement returns the value 0, the first database value:

```
SELECT NEXT_DATABASE( NULL ) FROM iq_dummy
```

The following statement returns NULL, indicating that there are no more databases on the server:

```
SELECT NEXT_DATABASE( 0 ) FROM iq_dummy
```

Usage

You can use **NEXT_DATABASE** to enumerate the databases running on a database server. To get the first database, pass NULL; to get each subsequent database, pass the previous return value. The function returns NULL when there are no more databases.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *COL_NAME Function [System]* on page 175
- *DB_ID Function [System]* on page 210
- *DB_NAME Function [System]* on page 211
- *DB_PROPERTY Function [System]* on page 212
- *OBJECT_ID Function [System]* on page 290
- *OBJECT_NAME Function [System]* on page 291

NEXT_HTTP_HEADER function [Web service]

Returns the next HTTP header name.

Syntax

```
NEXT_HTTP_HEADER( header-name )
```

Parameters

- ***header-name* –** The name of the previous request header. If header-name is NULL, this function returns the name of the first HTTP request header.

Returns

LONG VARCHAR.

Note: The result data type is a LONG VARCHAR. If you use NEXT_HTTP_HEADER in a SELECT INTO statement, you must have an Unstructured Data Analytics Option license or use CAST and set HTML_DECODE to the correct data type and size.

Remarks

This function is used to iterate over the HTTP request headers returning the next HTTP header name. Calling it with NULL causes it to return the name of the first header. Subsequent headers are retrieved by passing the name of the previous header to the function. This function returns NULL when called with the name of the last header, or when not called from a web service.

Calling this function repeatedly returns all the header fields exactly once, but not necessarily in the order they appear in the HTTP request.

Standards and compatibility

- **SQL/2008 –** Vendor extension.

Example

When used within a stored procedure that is called by an HTTP web service, the following example displays the name and values of the HTTP request headers in the database server messages window.

```
BEGIN
  declare header_name long varchar;
  declare header_value long varchar;
  set header_name  = NULL;
header_loop:
  LOOP
    SET header_name = NEXT_HTTP_HEADER( header_name );
    IF header_name IS NULL THEN
      LEAVE header_loop
    END IF;
    SET header_value = HTTP_HEADER( header_name );
    MESSAGE 'HEADER: ', header_name, '=',
            header_value TO CONSOLE;
  END LOOP;
END;
```

NEXT_HTTP_VARIABLE function [Web service]

Returns the next HTTP variable name.

Syntax

```
NEXT_HTTP_VARIABLE( var-name )
```

Parameters

- ***var-name*** **–** The name of the previous variable. If *var-name* is NULL, this function returns the name of the first HTTP variable.

Returns

LONG VARCHAR.

Note: The result data type is a LONG VARCHAR. If you use NEXT_HTTP_VARIABLE in a SELECT INTO statement, you must have an Unstructured Data Analytics Option license or use CAST and set NEXT_HTTP_HEADER to the correct data type and size.

Remarks

This function iterates over the HTTP variables included within a request. Calling it with NULL causes it to return the name of the first variable. Subsequent variables are retrieved by passing the function the name of the previous variable. This function returns NULL when called with the name of the final variable or when not called from a web service.

Calling this function repeatedly returns all the variables exactly once, but not necessarily in the order they appear in the HTTP request. The variables url or url1, url2, ..., url10 are included if URL PATH is set to ON or ELEMENTS, respectively.

Standards and compatibility

- **SQL/2008 –** Vendor extension.

Example

When used within a stored procedure that is called by an HTTP web service, the following example returns the name of the first HTTP variable.

```
BEGIN
DECLARE variable_name LONG VARCHAR;
DECLARE variable_value LONG VARCHAR;
SET variable_name = NULL;
SET variable_name = NEXT_HTTP_VARIABLE( variable_name );
SET variable_value = HTTP_VARIABLE( variable_name );
END;
```

NOW Function [Date and Time]

Returns the current date and time. This is the historical syntax for **CURRENT TIMESTAMP**.

Syntax

```
NOW ( * )
```

Returns

TIMESTAMP

Example

The following statement returns the current date and time.

```
SELECT NOW(*) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

NTILE Function [Analytical]

Distributes query results into a specified number of buckets and assigns the bucket number to each row in the bucket.

Syntax

```
NTILE ( expression1 )
OVER ( ORDER BY expression2 [ ASC | DESC ] )
```

Parameters

Parameter	Description
expression1	A constant integer from 1 to 32767, which specifies the number of buckets.
expression2	A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Example

The following example uses the **NTILE** function to determine the sale status of car dealers. The dealers are divided into four groups based on the number of cars each dealer sold. The dealers with ntile = 1 are in the top 25% for car sales.

```
SELECT dealer_name, sales,
NTILE(4) OVER ( ORDER BY sales DESC )
FROM carSales;
```

```
dealer_name          sales       ntile
Boston               1000        1
Worcester            950         1
Providence           950         1
SF                   940         1
Lowell               900         2
Seattle              900         2
Natick               870         2
New Haven            850         2
Portland             800         3
Houston              780         3
Hartford             780         3
Dublin               750         3
Austin               650         4
Dallas               640         4
Dover                600         4
```

To find the top 10% of car dealers by sales, you specify **NTILE(10)** in the example **SELECT** statement. Similarly, to find the top 50% of car dealers by sales, specify **NTILE(2)**.

Usage

NTILE is a rank analytical function that distributes query results into a specified number of buckets and assigns the bucket number to each row in the bucket. You can divide a result set into one-hundredths (percentiles), tenths (deciles), fourths (quartiles), or other numbers of groupings.

NTILE requires an **OVER (ORDER BY)** clause. The **ORDER BY** clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the **OVER** clause and is not an **ORDER BY** for the **SELECT**. No aggregation functions in the rank query are allowed to specify **DISTINCT**.

The **OVER** clause indicates that the function operates on a query result set. The result set is the rows that are returned after the **FROM**, **WHERE**, **GROUP BY**, and **HAVING** clauses have all been evaluated. The **OVER** clause defines the data set of the rows to include in the computation of the rank analytical function.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

NTILE is allowed only in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. **NTILE** can be in a view or a union. The **NTILE** function cannot be used in a subquery, a **HAVING** clause, or in the select list of an **UPDATE** or **DELETE** statement. Only one **NTILE** function is allowed per query.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere

See also

- *PERCENTILE_CONT Function [Analytical]* on page 296
- *PERCENTILE_DISC Function [Analytical]* on page 298
- *YEAR Function [Date and Time]* on page 380

NULLIF Function [Miscellaneous]

Provides an abbreviated **CASE** expression by comparing expressions.

Syntax

```
NULLIF ( expression1, expression2 )
```

Parameters

Parameter	Description
expression1	An expression to be compared.
expression2	An expression to be compared.

Returns

Data type of the first argument.

Examples

The following statement returns a:

```
SELECT NULLIF( 'a', 'b' ) FROM iq_dummy
```

The following statement returns NULL:

```
SELECT NULLIF( 'a', 'a' ) FROM iq_dummy
```

Usage

NULLIF compares the values of the two expressions.

If the first expression equals the second expression, **NULLIF** returns NULL.

If the first expression does not equal the second expression, or if the second expression is NULL, **NULLIF** returns the first expression.

The **NULLIF** function provides a short way to write some **CASE** expressions. **NULLIF** is equivalent to:

```
CASE WHEN expression1 = expression2 THEN NULL
ELSE expression1 END
```

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *CASE Expressions* on page 35
- *NULLIF Function for Abbreviated CASE Expressions* on page 36

NUMBER Function [Miscellaneous]

Generates numbers starting at 1 for each successive row in the results of the query.

Syntax

```
NUMBER ( * )
```

Returns

INT

Usage

Use the **NUMBER** function only in a select list or a **SET** clause of an **UPDATE** statement. For example, the following statement updates each row of the `seq_id` column with a number 1 greater than the previous row. The number is applied in the order specified by the **ORDER BY** clause.

```
update empl
set seq_id = number(*)
order by empl_id
```

In an **UPDATE** statement, if the **NUMBER**(*) function is used in the **SET** clause and the **FROM** clause specifies a one-to-many join, **NUMBER**(*) generates unique numbers that increase, but may not increment sequentially due to row elimination.

NUMBER can also be used to generate primary keys when using the **INSERT** from **SELECT** statement, although using **IDENTITY/AUTOINCREMENT** is a preferred mechanism for generating sequential primary keys.

Note: A syntax error is generated if you use **NUMBER** in a **DELETE** statement, **WHERE** clause, **HAVING** clause, **ORDER BY** clause, subquery, query involving aggregation, any constraint, **GROUP BY**, **DISTINCT**, a query containing **UNION ALL**, or a derived table.

Example

The following statement returns this numbered list:

number(*)
1
2
3
4

number(*)
5

```
SELECT NUMBER( * )
FROM Departments
WHERE DepartmentID > 10
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

OBJECT_ID Function [System]

Returns the object ID.

Syntax

```
OBJECT_ID ( object-name )
```

Parameters

Parameter	Description
object-name	The name of the object.

Examples

The following statement returns the object ID 100209 of the *Customers* table:

```
SELECT OBJECT_ID ('CUSTOMERS') FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ.

See also

- *COL_NAME Function [System]* on page 175
- *DB_ID Function [System]* on page 210
- *DB_NAME Function [System]* on page 211
- *DB_PROPERTY Function [System]* on page 212
- *NEXT_DATABASE Function [System]* on page 283
- *OBJECT_NAME Function [System]* on page 291
- *INDEX_COL Function [System]* on page 246

OBJECT_NAME Function [System]

Returns the object name.

Syntax

```
OBJECT_NAME ( object-id [ , database-id ] )
```

Parameters

Parameter	Description
object-id	The object ID.
database-id	The database ID.

Examples

The following statement returns the name “customer:”

```
SELECT OBJECT_NAME ( 100209 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ.

See also

- *BIT_LENGTH Function [String]* on page 165
- *BYTE_LENGTH Function [String]* on page 166
- *CHAR_LENGTH Function [String]* on page 171
- *COL_LENGTH Function [System]* on page 175
- *DATALENGTH Function [System]* on page 189
- *LEN Function [String]* on page 260
- *LENGTH Function [String]* on page 261
- *OCTET_LENGTH Function [String]* on page 292
- *STR_REPLACE Function [String]* on page 352
- *COL_NAME Function [System]* on page 175
- *DB_ID Function [System]* on page 210
- *DB_NAME Function [System]* on page 211
- *DB_PROPERTY Function [System]* on page 212
- *NEXT_DATABASE Function [System]* on page 283
- *OBJECT_ID Function [System]* on page 290

OCTET_LENGTH Function [String]

Returns an unsigned 64-bit value containing the byte length of the column parameter.

Syntax

```
OCTET_LENGTH( column-name )
```

Parameters

Parameter	Description
column-name	The name of a column.

Usage

The return value of a NULL argument is NULL.

The **OCTET_LENGTH** function supports all SAP Sybase IQ data types.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *OCTET_LENGTH Function* in *Unstructured Data Analytics*.

Standards and Compatibility

- Sybase—Not supported by SQL Anywhere or Adaptive Server Enterprise.

See also

- *BIT_LENGTH Function [String]* on page 165
- *BYTE_LENGTH Function [String]* on page 166
- *CHAR_LENGTH Function [String]* on page 171
- *COL_LENGTH Function [System]* on page 175
- *DATALENGTH Function [System]* on page 189
- *LEN Function [String]* on page 260
- *LENGTH Function [String]* on page 261
- *OBJECT_NAME Function [System]* on page 291
- *STR_REPLACE Function [String]* on page 352

PATINDEX Function [String]

Returns the starting position of the first occurrence of a specified pattern.

Syntax

```
PATINDEX ( '%pattern%', string-expression )
```

Parameters

Parameter	Description
pattern	The pattern for which you are searching. This string is limited to 126 bytes for patterns with wildcards. If the leading percent wildcard is omitted, **PATINDEX** returns one (1) if the pattern occurs at the beginning of the string, and zero if not. If *pattern* starts with a percent wildcard, then the two leading percent wildcards are treated as one. Patterns without wildcards (percent % or underscore _) can be up to 255 bytes in length.
string-expression	The string to be searched for the pattern.

Returns
INT

Examples
The following statement returns the value 2:

```
SELECT PATINDEX( '%hoco%', 'chocolate' ) FROM iq_dummy
```

The following statement returns the value 11:

```
SELECT PATINDEX ('%4_5_', '0a1A 2a3A 4a5A') FROM iq_dummy
```

Usage
PATINDEX returns the starting position of the first occurrence of the pattern. If the string being searched contains more than one instance of the string pattern, **PATINDEX** returns only the position of the first instance.

The pattern uses the same wildcards as the **LIKE** comparison. This table lists the pattern wildcards.

Table 55. PATINDEX pattern wildcards

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

If the pattern is not found, **PATINDEX** returns zero (0).

Searching for a pattern longer than 126 bytes returns NULL.

Searching for a zero-length string returns 1.

If any of the arguments is NULL, the result is zero (0).

All the positions or offsets, returned or specified, in the **PATINDEX** function are always character offsets and may be different from the byte offset for multibyte data.

PATINDEX returns a 32 bit unsigned integer position for `CHAR` and `VARCHAR` columns.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *PATINDEX Function* in *Unstructured Data Analytics*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *LIKE Conditions* on page 47
- *LOCATE Function [String]* on page 266

PERCENT_RANK Function [Analytical]

Computes the (fractional) position of one row returned from a query with respect to the other rows returned by the query, as defined by the **ORDER BY** clause.

Returns a decimal value between 0 and 1.

Syntax

```
PERCENT_RANK () OVER ( ORDER BY expression [ ASC | DESC ] )
```

Parameters

Parameter	Description
expression	A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Returns

The **PERCENT_RANK** function returns a DOUBLE value between 0 and 1.

Example

The following statement illustrates the use of the **PERCENT_RANK** function:

```
SELECT s_suppkey, SUM(s_acctBal) AS sum_acctBal,
PERCENT_RANK() OVER ( ORDER BY SUM(s_acctBal) DESC )
AS percent_rank_all FROM supplier GROUP BY s_suppkey;

s_suppkey        sum_acctBal       percent_rank_all
supplier#011     200000            0
supplier#002     200000            0
supplier#013     123000            0.3333
supplier#004     110000            0.5
supplier#035     110000            0.5
supplier#006     50000             0.8333
supplier#021     10000             1
```

Usage

PERCENT_RANK is a rank analytical function. The percent rank of a row R is defined as the rank of a row in the groups specified in the **OVER** clause minus one divided by the number of total rows in the groups specified in the **OVER** clause minus one. **PERCENT_RANK** returns a value between 0 and 1. The first row has a percent rank of zero.

The **PERCENT_RANK** of a row is calculated as

```
(Rx - 1) / (NtotalRow - 1)
```

where *Rx* is the rank position of a row in the group and *NtotalRow* is the total number of rows in the group specified by the **OVER** clause.

PERCENT_RANK requires an **OVER (ORDER BY)** clause. The **ORDER BY** clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the **OVER** clause and is not an **ORDER BY** for the **SELECT**. No aggregation functions in the rank query are allowed to specify **DISTINCT**.

The **OVER** clause indicates that the function operates on a query result set. The result set is the rows that are returned after the **FROM**, **WHERE**, **GROUP BY**, and **HAVING** clauses have all been evaluated. The **OVER** clause defines the data set of the rows to include in the computation of the rank analytical function.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

PERCENT_RANK is allowed only in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. **PERCENT_RANK** can be in a view or a union. The **PERCENT_RANK** function cannot be used in a subquery, a **HAVING** clause, or in the select list of an **UPDATE** or **DELETE** statement. Only one rank analytical function is allowed per query.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

PERCENTILE_CONT Function [Analytical]

Given a percentile, returns the value that corresponds to that percentile. Assumes a continuous distribution data model.

Note: If you are simply trying to compute a percentile, use the **NTILE** function instead, with a value of 100.

Syntax

```
PERCENTILE_CONT ( expression1 )
WITHIN GROUP  ( ORDER BY expression2 [ ASC | DESC ] )
```

Parameters

Parameter	Description
expression1	A constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, a "wrong argument for percentile" error is returned. If the argument value is less than 0 or greater than 1, a "data value out of range" error is returned
expression2	A sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

Example

The following example uses the **PERCENTILE_CONT** function to determine the 10th percentile value for car sales in a region.

The following data set is used in the example:

```
sales        region        dealer_name
900          Northeast     Boston
800          Northeast     Worcester
800          Northeast     Providence
700          Northeast     Lowell
540          Northeast     Natick
500          Northeast     New Haven
450          Northeast     Hartford
800          Northwest     SF
600          Northwest     Seattle
500          Northwest     Portland
400          Northwest     Dublin
500          South         Houston
400          South         Austin
```

```
300         South         Dallas
200         South         Dover
```

The following **SELECT** statement contains the **PERCENTILE_CONT** function:

```
SELECT region, PERCENTILE_CONT(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

The result of the **SELECT** statement lists the 10th percentile value for car sales in a region:

```
region           percentile_cont
Northeast        840
Northwest        740
South            470
```

Usage

The inverse distribution analytical functions return a k-th percentile value, which can be used to help establish a threshold acceptance value for a set of data. The function **PERCENTILE_CONT** takes a percentile value as the function argument, and operates on a group of data specified in the **WITHIN GROUP** clause, or operates on the entire data set. The function returns one value per group. If the **GROUP BY** column from the query is not present, the result is a single row. The data type of the results is the same as the data type of its **ORDER BY** item specified in the **WITHIN GROUP** clause. The data type of the **ORDER BY** expression for **PERCENTILE_CONT** must be numeric.

PERCENTILE_CONT requires a **WITHIN GROUP (ORDER BY)** clause.

The **ORDER BY** clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. For the **PERCENTILE_CONT** function, the data type of this expression must be numeric. This **ORDER BY** clause is used only within the **WITHIN GROUP** clause and is not an **ORDER BY** for the **SELECT**.

The **WITHIN GROUP** clause distributes the query result into an ordered data set from which the function calculates a result. The **WITHIN GROUP** clause must contain a single sort item. If the **WITHIN GROUP** clause contains more or less than one sort item, an error is reported.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

The **PERCENTILE_CONT** function is allowed in a subquery, a **HAVING** clause, a view, or a union. **PERCENTILE_CONT** can be used anywhere the simple nonanalytical aggregate functions are used. The **PERCENTILE_CONT** function ignores the NULL value in the data set.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

- *NTILE Function [Analytical]* on page 286
- *PERCENTILE_DISC Function [Analytical]* on page 298

PERCENTILE_DISC Function [Analytical]

Given a percentile, returns the value that corresponds to that percentile. Assumes a discrete distribution data model.

Note: If you are simply trying to compute a percentile, use the **NTILE** function instead, with a value of 100.

Syntax

```
PERCENTILE_DISC ( expression1 )
WITHIN GROUP  ( ORDER BY expression2 [ ASC | DESC ] )
```

Parameters

Parameter	Description
expression1	A constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, then a "wrong argument for percentile" error is returned. If the argument value is less than 0 or greater than 1, then a "data value out of range" error is returned.
expression2	A sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

Example

The following example uses the **PERCENTILE_DISC** function to determine the 10th percentile value for car sales in a region.

The following data set is used in the example:

```
sales       region       dealer_name
900         Northeast    Boston
800         Northeast    Worcester
800         Northeast    Providence
700         Northeast    Lowell
540         Northeast    Natick
500         Northeast    New Haven
450         Northeast    Hartford
800         Northwest    SF
600         Northwest    Seattle
500         Northwest    Portland
```

```
400         Northwest     Dublin
500         South         Houston
400         South         Austin
300         South         Dallas
200         South         Dover
```

The following **SELECT** statement contains the **PERCENTILE_DISC** function:

```
SELECT region, PERCENTILE_DISC(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

The result of the **SELECT** statement lists the 10th percentile value for car sales in a region:

```
region           percentile_cont
Northeast        900
Northwest        800
South            500
```

Usage

The inverse distribution analytical functions return a k-th percentile value, which can be used to help establish a threshold acceptance value for a set of data. The function **PERCENTILE_DISC** takes a percentile value as the function argument and operates on a group of data specified in the **WITHIN GROUP** clause or operates on the entire data set. The function returns one value per group. If the **GROUP BY** column from the query is not present, the result is a single row. The data type of the results is the same as the data type of its **ORDER BY** item specified in the **WITHIN GROUP** clause. **PERCENTILE_DISC** supports all data types that can be sorted in SAP Sybase IQ.

PERCENTILE_DISC requires a **WITHIN GROUP (ORDER BY)** clause.

The **ORDER BY** clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the **WITHIN GROUP** clause and is not an **ORDER BY** for the **SELECT**.

The **WITHIN GROUP** clause distributes the query result into an ordered data set from which the function calculates a result. The **WITHIN GROUP** clause must contain a single sort item. If the **WITHIN GROUP** clause contains more or less than one sort item, an error is reported.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

The **PERCENTILE_DISC** function is allowed in a subquery, a **HAVING** clause, a view, or a union. **PERCENTILE_DISC** can be used anywhere the simple nonanalytical aggregate functions are used. The **PERCENTILE_DISC** function ignores the NULL value in the data set.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

- *NTILE Function [Analytical]* on page 286
- *PERCENTILE_CONT Function [Analytical]* on page 296

PI Function [Numeric]

Returns the numeric value PI.

Syntax

```
PI ( * )
```

Returns

DOUBLE

Example

The following statement returns the value 3.141592653....

```
SELECT PI( * ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—The **PI()** function is supported in Adaptive Server Enterprise, but **PI(*)** is not.

POWER Function [Numeric]

Calculates one number raised to the power of another.

Syntax

```
POWER ( numeric-expression1, numeric-expression2 )
```

Parameters

Parameter	Description
numeric-expression1	The base.
numeric-expression2	The exponent.

Returns

DOUBLE

Example

The following statement returns the value 64:

```
SELECT Power( 2, 6 ) FROM iq_dummy
```

Usage
Raises *numeric-expression1* to the power *numeric-expresson2*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

PROPERTY Function [System]

Returns the value of the specified server-level property as a string.

Syntax

```
PROPERTY ( { property-id  |  property-name } )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Table 56.

Parameter	Description
property-id	An integer that is the property-number of the server-level property. This number can be determined from the **PROPERTY_NUMBER** function. The *property-id* is commonly used when looping through a set of properties.
property-name	A string giving the name of the property.

Returns
VARCHAR

Example
The following statement returns the name of the current database server:

```
SELECT PROPERTY( 'Name' ) FROM iq_dummy
```

Usage
Each property has both a number and a name, but the number is subject to change between versions, and should not be used as a reliable identifier for a given property.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Not supported by Adaptive Server Enterprise.

See also
- *PROPERTY_NAME Function [System]* on page 303
- *PROPERTY_NUMBER Function [System]* on page 303
- *CONNECTION_PROPERTY Function [System]* on page 176
- *Properties Available for the Server* on page 101
- *Properties Available for Each Database* on page 133
- *Connection Properties* on page 100

PROPERTY_DESCRIPTION Function [System]

Returns a description of a property.

Syntax

```
PROPERTY_DESCRIPTION ( { property-id | property-name } )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
property-id	An integer that is the property number of the property. This number can be determined from the **PROPERTY_NUMBER** function. The *property-id* is commonly used when looping through a set of properties.
property-name	A string giving the name of the property.

Returns

VARCHAR

Example

The following statement returns the description “Number of index insertions:”

```
SELECT PROPERTY_DESCRIPTION( 'IndAdd' ) FROM iq_dummy
```

Usage

Each property has both a number and a name, but the number is subject to change between releases, and should not be used as a reliable identifier for a given property.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Not supported by Adaptive Server Enterprise.

PROPERTY_NAME Function [System]

Returns the name of the property with the supplied property number.

Syntax

```
PROPERTY_NAME ( property-id )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
property-id	The property number of the property.

Returns

VARCHAR

Example

The following statement returns the property associated with property number 126. The actual property to which this refers changes from version to version.

```
SELECT PROPERTY_NAME( 126 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *PROPERTY Function [System]* on page 301
- *PROPERTY_NUMBER Function [System]* on page 303
- *CONNECTION_PROPERTY Function [System]* on page 176
- *Properties Available for the Server* on page 101
- *Properties Available for Each Database* on page 133
- *Connection Properties* on page 100

PROPERTY_NUMBER Function [System]

Returns the property number of the property with the supplied property name.

Syntax

```
PROPERTY_NUMBER ( property-name )
```

Note: CIS functional compensation performance considerations apply.

Parameters

Parameter	Description
property-name	A property name.

Returns

INT

Example

The following statement returns an integer value. The actual value changes from version to version.

```
SELECT PROPERTY_NUMBER( 'PAGESIZE' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *PROPERTY Function [System]* on page 301
- *PROPERTY_NAME Function [System]* on page 303
- *CONNECTION_PROPERTY Function [System]* on page 176
- *Properties Available for the Server* on page 101
- *Properties Available for Each Database* on page 133
- *Connection Properties* on page 100

QUARTER Function [Date and Time]

Returns a number indicating the quarter of the year from the supplied date expression.

Syntax

```
QUARTER( date-expression )
```

Parameters

Parameter	Description
date-expression	A date.

Returns

INT

Example

With the **DATE_ORDER** option set to the default of *ymd*, the following statement returns the value 2:

```
SELECT QUARTER ( '1987/05/02' ) FROM iq_dummy
```

Usage

This table lists the dates in the quarters of the year.

Table 57. Values of quarter of the year

Quarter	Period (inclusive)
1	January 1 to March 31
2	April 1 to June 30
3	July 1 to September 30
4	October 1 to December 31

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

RADIANS Function [Numeric]

Converts a number from degrees to radians.

Syntax

RADIANS (*numeric-expression*)

Parameters

Parameter	Description
numeric-expression	A number, in degrees. This angle is converted to radians

Returns

DOUBLE

Example

The following statement returns a value of approximately 0.5236:

```
SELECT RADIANS( 30 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

RAND Function [Numeric]

Returns a `DOUBLE` precision, random number x, where 0 <= x <1, with an optional seed.

Syntax

```
RAND ( [ integer-expression ] )
```

Parameters

Parameter	Description
integer-expression	The optional seed used to create a random number. This argument allows you to create repeatable random number sequences.

Returns

DOUBLE

Examples

The following statement returns a 5% sampling of a table:

```
SELECT AVG(table1.number_of_cars), AVG(table1.number_of_tvs)FROM
table1 WHERE RAND(ROWID(table1)) < .05 and table1.income < 50000;
```

The following statement returns a value of approximately 941392926249216914:

```
SELECT RAND( 4 ) FROM iq_dummy
```

Usage

If **RAND** is called with a **FROM** clause and an argument in a query containing only tables in IQ stores, the function returns an arbitrary but repeatable value.

When no argument is called, **RAND** is a non-deterministic function. Successive calls to **RAND** might return different values. The query optimizer does not cache the results of the **RAND** function

Note: The values returned by **RAND** vary depending on whether you use a **FROM** clause or not and whether the referenced table was created in `SYSTEM` or in an IQ store.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Compatible with Adaptive Server Enterprise.

RANK Function [Analytical]

Ranks items in a group.

Syntax

```
RANK () OVER ( [ PARTITION BY ] ORDER BY expression [ ASC | DESC ] )
```

Parameters

Parameter	Description
expression	A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Returns

INTEGER

Example

This statement illustrates the use of the **RANK** function:

```
SELECT Surname, Sex, Salary, RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) AS RANK FROM Employees
WHERE State IN ('CA', 'AZ') AND DepartmentID IN (200, 300)
ORDER BY Sex, Salary DESC;
```

The results from the above query:

```
Surname          Sex      Salary     RANK
-------          ---      ------     ----
Savarino          F        72300.000  1
Jordan            F        51432.000  2
Clark             F        45000.000  3
Coleman           M        42300.000  1
Overbey           M        39300.000  2
```

Usage

RANK is a rank analytical function. The rank of row R is defined as the number of rows that precede R and are not peers of R. If two or more rows are not distinct within the groups specified in the **OVER** clause or distinct over the entire result set, then there are one or more gaps in the sequential rank numbering. The difference between **RANK** and **DENSE_RANK** is that **DENSE_RANK** leaves no gap in the ranking sequence when there is a tie. **RANK** leaves a gap when there is a tie.

RANK requires an **OVER (ORDER BY)** clause. The **ORDER BY** clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This **ORDER BY** clause is used only within the **OVER** clause and is not an **ORDER BY** for the **SELECT**. No aggregation functions in the rank query are allowed to specify DISTINCT.

The **PARTITION BY** window partitioning clause in the **OVER (ORDER BY)** clause is optional.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

The **OVER** clause indicates that the function operates on a query result set. The result set is the rows that are returned after the **FROM**, **WHERE**, **GROUP BY**, and **HAVING** clauses have all been evaluated. The **OVER** clause defines the data set of the rows to include in the computation of the rank analytical function.

RANK is allowed only in the select list of a **SELECT** or **INSERT** statement or in the **ORDER BY** clause of the **SELECT** statement. **RANK** can be in a view or a union. The **RANK** function cannot be used in a subquery, a **HAVING** clause, or in the select list of an **UPDATE** or **DELETE** statement. Only one rank analytical function is allowed per query.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

- *DENSE_RANK Function [Analytical]* on page 213

REGR_AVGX Function [Aggregate]

Computes the average of the independent variable of the regression line.

Syntax 1

```
REGR_AVGX (dependent-expression, independent-expression)
```

Syntax 2

```
REGR_AVGX (dependent-expression, independent-expression)
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns
DOUBLE

Usage
This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_AVGX** returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where x represents the independent-expression:

```
AVG (x)
```

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example
The following example calculates the average of the dependent variable, employee age:

```
SELECT REGR_AVGX( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

See also
- *Windowing Aggregate Function Usage* on page 83

REGR_AVGY Function [Aggregate]

Computes the average of the dependent variable of the regression line.

Syntax 1

```
REGR_AVGY (dependent-expression, independent-expression)
```

Syntax 2

```
REGR_AVGY (dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns
DOUBLE

Usage
This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_AVGY** returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where y represents the dependent-expression:

```
AVG(y)
```

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example
The following example calculates the average of the independent variable, employee salary. This function returns the value 49988.6232:

```
SELECT REGR_AVGY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

REGR_COUNT Function [Aggregate]

Returns an integer that represents the number of non-NULL number pairs used to fit the regression line.

Syntax 1

```
REGR_COUNT(dependent-expression, independent-expression)
```

Syntax 2

```
REGR_COUNT(dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

INTEGER

Usage

This function returns an UNSIGNED BIGINT as the result.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns a value that indicates the number of non-NULL pairs that were used to fit the regression line. This function returns the value 75:

```
SELECT REGR_COUNT( Salary, ( YEAR( NOW() ) -
YEAR( BirthDate ) ) )FROM Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

See also

- *Windowing Aggregate Function Usage* on page 83

REGR_INTERCEPT Function [Aggregate]

Computes the y-intercept of the linear regression line that best fits the dependent and independent variables.

Syntax 1

```
REGR_INTERCEPT(dependent-expression, independent-expression)
```

Syntax 2

```
REGR_INTERCEPT(dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns
DOUBLE

Usage
This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, **REGR_INTERCEPT** returns NULL.

The function is applied to the set of (dependent-expression and *independent-expression*) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the *independent-expression*:

```
AVG(y) - REGR_SLOPE(y, x) * AVG(x)
```

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns the value 1874.5805688517603:

```
SELECT REGR_INTERCEPT( Salary, ( YEAR( NOW() ) -
YEAR( BirthDate ) ) )FROM Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

See also

- *Windowing Aggregate Function Usage* on page 83

REGR_R2 Function [Aggregate]

Computes the coefficient of determination (also referred to as R-squared or the goodness-of-fit statistic) for the regression line.

Syntax 1

REGR_R2(*dependent-expression*, *independent-expression*)

Syntax 2

REGR_R2(*dependent-expression*, *independent-expression*)

OVER (*window-spec*)

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_R2** returns NULL.

REGR_R2 is applied to the set of (dependent-expression and *independent-expression*) pairs after eliminating all pairs for which either dependent-expression or *independent-expression* is NULL. SAP Sybase IQ then applies the following algorithm:

- **REGR_R2** calculates VAR_POP(x) and returns NULL if VAR_POP(x) = 0; otherwise, it calculates VAR_POP(y) and returns the value 1 if VAR_POP(y) = 0.
- If neither VAR_POP(x) or VAR_POP(y) is zero, the return value is POWER(CORR(y,x), 2)

where y represents the *dependent-expression* and x represents the *independent-expression*.

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. **DISTINCT** is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns the value 0.19379959710325653:

```
SELECT REGR_R2( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

See also

- *Windowing Aggregate Function Usage* on page 83

REGR_SLOPE Function [Aggregate]

Computes the slope of the linear regression line, fitted to non-NULL pairs.

Syntax 1

```
REGR_SLOPE(dependent-expression, independent-expression)
```

Syntax 2

```
REGR_SLOPE(dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns
DOUBLE

Usage
This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_SLOPE** returns NULL.

REGR_SLOPE is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the independent-expression:

```
COVAR_POP(x, y) / VAR_POP(y)
```

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example
The following example returns the value 935.3429749445614:

```
SELECT REGR_SLOPE( Salary, ( YEAR( NOW() ) -
YEAR( BirthDate ) ) )FROM Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

See also
- *Windowing Aggregate Function Usage* on page 83

REGR_SXX Function [Aggregate]
Computes the slope of the linear regression line, fitted to non-NULL pairs.

Syntax 1

```
REGR_SXX(dependent-expression, independent-expression)
```

Syntax 2

```
REGR_SXX(dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_SXX** returns NULL.

The function is applied to the set of (dependent-expression and *independent-expression*) pairs after eliminating all pairs for which either dependent-expression or *independent-expression* is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the *independent-expression*:

```
REGR_COUNT(y, x) * VAR_POP(x)
```

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns the value 5916.4800000000105:

```
SELECT REGR_SXX( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

See also

- *Windowing Aggregate Function Usage* on page 83

REGR_SXY Function [Aggregate]

Returns the sum of products of the dependent and independent variables. Use REGR_SXY to evaluate the statistical validity of a regression model.

Syntax 1

```
REGR_SXY(dependent-expression, independent-expression)
```

Syntax 2

```
REGR_SXY(dependent-expression, independent-expression)
```

```
OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then it returns NULL.

The function is applied to the set of (dependent-expression and *independent-expression*) pairs after eliminating all pairs for which either dependent-expression or *independent-expression* is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the *independent-expression*:

```
REGR_COUNT(x, y) * COVAR_POP(x,
y)
```

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns the value 5533938.004400015.

```
SELECT REGR_SXY( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

See also

- *Windowing Aggregate Function Usage* on page 83

REGR_SYY Function [Aggregate]

Returns values that can evaluate the statistical validity of a regression model.

Syntax 1

```
REGR_SYY(dependent-expression, independent-expression)
```

Syntax 2

```
REGR_SYY(dependent-expression, independent-expression)

OVER (window-spec)
```

Parameters

Parameter	Description
dependent-expression	The variable that is affected by the independent variable.
independent-expression	The variable that influences the outcome.

Returns

DOUBLE

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_SYY** returns NULL.

The function is applied to the set of (*dependent-expression* and independent-expression) pairs after eliminating all pairs for which either *dependent-expression* or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where y represents the *dependent-expression* and x represents the independent-expression:

```
REGR_COUNT(x, y) * VAR_POP(y)
```

Note: ROLLUP and CUBE are not supported in the **GROUP BY** clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a **SELECT** statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

Example

The following example returns the value 26, 708, 672,843.3002:

```
SELECT REGR_SYY( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )FROM
Employees;
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T612.
- Sybase—Compatible with SQL Anywhere.

See also

- *Windowing Aggregate Function Usage* on page 83

REMAINDER Function [Numeric]

Returns the remainder when one whole number is divided by another.

Syntax

```
REMAINDER ( dividend, divisor )
```

Parameters

Parameter	Description
dividend	The dividend, or numerator of the division.
divisor	The divisor, or denominator of the division.

Returns

INTEGER

NUMERIC

Example

The following statement returns the value 2:

```
SELECT REMAINDER( 5, 3 ) FROM iq_dummy
```

Usage

REMAINDER is the same as the **MOD** function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported in Adaptive Server Enterprise. The % (modulo) operator and the division operator can be used to produce a remainder.

See also

- *MOD Function [Numeric]* on page 276

REPEAT Function [String]

Concatenates a string a specified number of times.

Syntax

```
REPEAT ( string-expression, integer-expression )
```

Parameters

Parameter	Description
string-expression	The string to be repeated.
integer-expression	The number of times the string is to be repeated. If *integer-expression* is negative, an empty string is returned.

Returns

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **REPEAT** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set REPEAT to the correct data type and size.

Example

The following statement returns the value “repeatrepeatrepeat:”

```
SELECT REPEAT( 'repeat', 3 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Not supported in Adaptive Server Enterprise, but **REPLICATE** provides the same capabilities.

See also

- *REPLACE Function [String]* on page 321
- *REPLICATE Function [String]* on page 323

REPLACE Function [String]

Replaces all occurrences of a substring with another substring.

Syntax

```
REPLACE ( original-string, search-string, replace-string )
```

Parameters

If any argument is NULL, the function returns NULL.

Parameter	Description
original-string	The string to be searched. This string can be any length.
search-string	The string to be searched for and replaced with *replace-string*. This string is limited to 255 bytes. If *search-string* is an empty string, the original string is returned unchanged.
replace-string	The replacement string, which replaces *search-string*. This can be any length. If *replace-string* is an empty string, all occurrences of *search-string* are deleted.

Returns

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **REPLACE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **REPLACE** to the correct data type and size.

Examples

The following statement returns the value “xx.def.xx.ghi:”

```
SELECT REPLACE( 'abc.def.abc.ghi', 'abc', 'xx' ) FROM iq_dummy
```

The following statement generates a result set containing **ALTER PROCEDURE** statements which, when executed, repair stored procedures that reference a table that has been renamed. (To be useful, the table name must be unique.)

```
SELECT REPLACE(
  replace(proc_defn,'OldTableName','NewTableName'),
  'create procedure',
  'alter procedure')
FROM SYS.SYSPROCEDURE
WHERE proc_defn LIKE '%OldTableName%'
```

Use a separator other than the comma for the **LIST** function:

```
SELECT REPLACE( list( table_id ), ',', '--')
FROM  SYS.ISYSTAB
WHERE table_id <= 5
```

Usage

The result data type of a **REPLACE** function is a `LONG VARCHAR`. If you use **REPLACE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license, or use **CAST** and set **REPLACE** to the correct data type and size.

There are two ways to work around this issue:

- Declare a local temporary table, then perform an **INSERT**:

  ```
  DECLARE local temporary table #mytable
    (name_column char(10)) on commit preserve rows;
  INSERT INTO #mytable SELECT REPLACE(name,'0','1')   FROM
  dummy_table01;
  ```
- Use **CAST**:

  ```
  SELECT CAST(replace(name, '0', '1') AS Char(10)) into #mytable
  from dummy_table01;
  ```

If you need to control the width of the resulting column when *replace-string* is wider than *search-string*, use the **CAST** function. For example,

```
CREATE TABLE aa(a CHAR(5));
INSERT INTO aa VALUES(‘CCCCC’);
COMMIT;
SELECT a, CAST(REPLACE(a,’C’,’ZZ’) AS CHAR(5)) FROM aa;
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *CAST Function [Data Type Conversion]* on page 167
- *CONVERT Function [Data Type Conversion]* on page 178
- *HOURS Function [Date and Time]* on page 235
- *MINUTES Function [Date and Time]* on page 275

- *MONTHS Function [Date and Time]* on page 278
- *SECOND Function [Date and Time]* on page 331
- *WEEKS Function [Date and Time]* on page 375
- *YEAR Function [Date and Time]* on page 380
- *YEARS Function [Date and Time]* on page 381
- *LCASE Function [String]* on page 256
- *LEFT Function [String]* on page 259
- *LOWER Function [String]* on page 269
- *REVERSE Function [String]* on page 324
- *RIGHT Function [String]* on page 325
- *UCASE Function [String]* on page 366
- *UPPER Function [String]* on page 367
- *REPEAT Function [String]* on page 320
- *REPLICATE Function [String]* on page 323

REPLICATE Function [String]

Concatenates a string a specified number of times.

Syntax

```
REPLICATE ( string-expression, integer-expression )
```

Parameters

Parameter	Description
string-expression	The string to be repeated.
integer-expression	The number of times the string is to be repeated.

Returns

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **REPLICATE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **REPLICATE** to the correct data type and size.

Example

The following statement returns the value “repeatrepeatrepeat:”

```
SELECT REPLICATE( 'repeat', 3 ) FROM iq_dummy
```

Usage

REPLICATE is the same as the **REPEAT** function.

Note: The result data type of a **REPLICATE** function is a `LONG VARCHAR`. If you use **REPLICATE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **REPLICATE** to the correct data type and size.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *REPEAT Function [String]* on page 320
- *REPLACE Function [String]* on page 321

REVERSE Function [String]

Takes one argument as an input of type `BINARY` or `STRING` and returns the specified string with characters listed in reverse order.

Syntax

```
REVERSE ( expression | uchar_expr )
```

Parameters

Parameter	Description
expression	A character or binary-type column name, variable, or constant expression of `CHAR`, `VARCHAR`, `NCHAR`, `NVARCHAR`, `BINARY`, or `VARBINARY` type.

Returns

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **REVERSE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **REVERSE** to the correct data type and size.

Example 1

```
select reverse("abcd")
----
dcba
```

Example 2

```
select reverse(0x12345000)
----------
0x00503412
```

Usage

- **REVERSE**, a string function, returns the reverse of expression.
- If expression is NULL, reverse returns NULL.
- Surrogate pairs are treated as indivisible and are not reversed.

Permissions

Any user can execute **REVERSE**.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.

See also

- *String Operators* on page 31
- *LCASE Function [String]* on page 256
- *LEFT Function [String]* on page 259
- *LOWER Function [String]* on page 269
- *REPLACE Function [String]* on page 321
- *RIGHT Function [String]* on page 325
- *UCASE Function [String]* on page 366
- *UPPER Function [String]* on page 367

RIGHT Function [String]

Returns the rightmost characters of a string.

Syntax

```
RIGHT ( string-expression, numeric-expression )
```

Parameters

Parameter	Description
string-expression	The string to be left-truncated.
numeric-expression	The number of characters at the end of the string to return.

Returns

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **RIGHT** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **RIGHT** to the correct data type and size.

Example

The following statement returns the value “olate:”

```
SELECT RIGHT( 'chocolate', 5 ) FROM iq_dummy
```

Usage

If the string contains multibyte characters, and the proper collation is being used, the number of bytes returned might be greater than the specified number of characters.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *LCASE Function [String]* on page 256
- *LEFT Function [String]* on page 259
- *LOWER Function [String]* on page 269
- *REPLACE Function [String]* on page 321
- *REVERSE Function [String]* on page 324
- *UCASE Function [String]* on page 366
- *UPPER Function [String]* on page 367

ROUND Function [Numeric]

Rounds the *numeric-expression* to the specified *integer-expression* number of places after the decimal point.

Syntax

```
ROUND ( numeric-expression, integer-expression )
```

Parameters

Parameter	Description
numeric-expression	The number, passed to the function, to be rounded.
integer-expression	A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative expression specifies the number of significant digits to the left of the decimal point at which to round.

Returns
NUMERIC

When `ROUND_TO_EVEN` database option is set ON, the **ROUND** function rounds data from a SAP Sybase IQ table half to the nearest even number to the integer-expression, matching the behavior of SQL Anywhere table data. When the option is set to OFF, the **ROUND** function rounds SAP Sybase IQ data half away from zero.

Examples
The following statement returns the value 123.200:

```
SELECT ROUND( 123.234, 1 ) FROM iq_dummy
```

Additional results of the **ROUND** function are shown in the following table:

Value	ROUND (Value)
123.4567	round (a.n,4)
123.4570	round (a.n,3)
123.4600	round (a.n,2)
123.5000	round (a.n,1)
123.0000	round (a.n, 0)
120.0000	round (a.n,-1)
100.0000	round (a.n,-2)
0.0000	round(a.n,-3)

In the following examples, the ROUND_TO_EVEN settings affects the value returned.

ROUND (Value)	ROUND_TO_EVEN ON	ROUND_TO_EVEN_OFF	Note
ROUND (convert (double, 123.45001), 1)	123.5	123.5	0.05001 is more than half of 0.1
ROUND (convert (double, 123.45000), 1)	123.4	123.5	0.0500 is half of 0.1

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *TRUNCNUM Function [Numeric]* on page 365

ROW_NUMBER Function [Analytical]

A ranking function that returns a unique row number for each row in a window partition, restarting the row numbering at the start of every window partition.

If no window partitions exist, the function numbers the rows in the result set from 1 to the cardinality of the table.

Syntax

```
ROW_NUMBER() OVER ([PARTITION BY window partition] ORDER BY window
ordering)
```

Parameters

Parameter	Description
window partition	(Optional) One or more value expressions separated by commas indicating how you want to divide the set of result rows.
window ordering	Defines the expressions for sorting rows within window partitions, if specified, or within the result set if you did not specify a window partition.

Usage

The **ROW_NUMBER** function requires an **OVER** (**ORDER_BY**) window specification. The window partitioning clause in the **OVER** (**ORDER_BY**) clause is optional. The **OVER** (**ORDER_BY**) clause must not contain a window frame **ROWS**/**RANGE** specification.

Example

The following example returns salary data from the Employees table, partitions the result set by department ID, and orders the data according to employee start date. The **ROW_NUMBER** function assigns each row a row number, and restarts the row numbering for each window partition:

```
SELECT DepartmentID dID, StartDate, Salary,
ROW_NUMBER()OVER(PARTITION BY dID ORDER BY StartDate) FROM  Employees
ORDER BY 1,2;
```

The returned result set is:

```
dID        StartDate    Salary      Row_number()
=========  ===========  ==========  =============
100         1986-10-14  42,998.000     1
100         1987-07-23  39,875.500     2
100         1988-03-23  37,400.000     3
100         1989-04-20  42,500.000     4
100         1990-01-15  42,100.000     5
200         1985-02-03  38,500.000     1
200         1987-02-19  39,300.000     2
200         1988-11-22  39,800.000     3
200         1989-06-01  34,892.000     4
200         1990-05-13  33,890.000     5
200         1990-07-11  37,803.000     6
```

Standards and Compatibility

- SQL—ISO/ANSI SQL compliant. SQL/OLAP feature T611.

ROWID Function [Miscellaneous]

Returns the internal row ID value for each row of the table.

Syntax

ROWID (*table-name*) …**FROM** *table-name*

Parameters

Parameter	Description
table-name	The name of the table. Specify the name of the table within the parentheses with either no quotes or with double quotes.

Returns

UNSIGNED BIGINT

Examples

The following statement returns the row ID values 1 through 10:

```
SELECT ROWID( “PRODUCTS” ) FROM PRODUCTS
```

rowid(Products)
1
2
3
.
.
.
10

The following statement returns the product ID and row ID value of all rows with a product ID value less than 400:

```
SELECT PRODUCTS.ID, ROWID ( PRODUCTS )
FROM PRODUCTS
WHERE PRODUCTS.ID < 400
```

ID	rowid (Products)
300	1
301	2
302	3

The following statement deletes all rows with row ID values greater than 50:

```
DELETE FROM PRODUCTS
WHERE ROWID ( PRODUCTS ) > 50
```

Usage

You can use the **ROWID** function with other clauses to manipulate specific rows of the table.

You must specify the **FROM** *table-name* clause.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

RTRIM Function [String]

Returns a string with trailing blanks removed.

Syntax

```
RTRIM ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string to be trimmed.

Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **RTRIM** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **RTRIM** to the correct data type and size.

Example

The following statement returns the string "Test Message" with all trailing blanks removed.

```
SELECT RTRIM( 'Test Message     ' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *LTRIM Function [String]* on page 270

SECOND Function [Date and Time]

Returns a number from 0 to 59 corresponding to the second component of the given date/time value.

Syntax

```
SECOND ( datetime-expression )
```

Parameters

Parameter	Description
datetime-expression	The date/time value.

Returns
SMALLINT

Example
The following statement returns the value 5:

```
SELECT SECOND( '1998-07-13 08:21:05' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *CAST Function [Data Type Conversion]* on page 167
- *CONVERT Function [Data Type Conversion]* on page 178
- *HOURS Function [Date and Time]* on page 235
- *MINUTES Function [Date and Time]* on page 275
- *MONTHS Function [Date and Time]* on page 278
- *REPLACE Function [String]* on page 321
- *WEEKS Function [Date and Time]* on page 375
- *YEAR Function [Date and Time]* on page 380
- *YEARS Function [Date and Time]* on page 381

SECONDS Function [Date and Time]

Returns the number of seconds since an arbitrary starting date and time, the number of seconds between two times, or adds an integer amount of seconds to a time.

Syntax

```
SECONDS ( datetime-expression
| datetime-expression, datetime-expression
| datetime-expression, integer-expression )
```

Parameters

Parameter	Description
datetime-expression	A date and time.
integer-expression	The number of seconds to be added to the date-time-expression. If integer-expression is negative, the appropriate number of minutes are subtracted from the date/time value. If you supply an integer expression, the datetime-expression must be explicitly cast as a datetime data type.

Returns

INTEGER

TIMESTAMP

Examples

The following statement returns the value 3600:

```
SELECT ( SECONDS( '1998-07-13 06:07:12' ) -
SECONDS( '1998-07-13 05:07:12' )) FROM iq_dummy
```

The following statement returns the value 14400, to signify the difference between the two times:

```
SELECT SECONDS( '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-05-12 21:05:12.000:

```
SELECT SECONDS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5) FROM iq_dummy
```

Usage

The second syntax returns the number of whole seconds from the first date/time to the second date/time. The number might be negative.

Standards and compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

SIGN Function [Numeric]

Returns the sign of a number.

Syntax

```
SIGN ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	The number for which the sign is to be returned.

Returns

SMALLINT

Example

The following statement returns the value -1:

```
SELECT SIGN( -550 ) FROM iq_dummy
```

Usage

For negative numbers, **SIGN** returns -1.

For zero, **SIGN** returns 0.

For positive numbers, **SIGN** returns 1.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

SIMILAR Function [String]

Returns an integer between 0 and 100 representing the similarity between two strings.

Syntax

```
SIMILAR ( string-expression1, string-expression2 )
```

Parameters

Parameter	Description
string-expression1	The first string to be compared.
string-expression2	The second string to be compared.

Returns

SMALLINT

Example

The following statement returns the value 75:

```
SELECT SIMILAR( 'toast', 'coast' ) FROM iq_dummy
```

This signifies that the two values are 75% similar.

Usage

The function returns an integer between 0 and 100 representing the similarity between the two strings. The result can be interpreted as the percentage of characters matched between the two strings. A value of 100 indicates that the two strings are identical.

This function can be used to correct a list of names (such as customers). Some customers might have been added to the list more than once with slightly different names. Join the table to itself and produce a report of all similarities greater than 90 percent but less than 100 percent.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

SIN Function [Numeric]

Returns the sine of a number, expressed in radians.

Syntax

```
SIN ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	The angle, in radians.

Returns

DOUBLE

Example

The following statement returns the value 0.496880:

```
SELECT SIN( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *COS Function [Numeric]* on page 183
- *ATAN2 Function [Numeric]* on page 162
- *ATAN Function [Numeric]* on page 161
- *ASIN Function [Numeric]* on page 160

- *ACOS Function [Numeric]* on page 158
- *COT Function [Numeric]* on page 184
- *TAN Function [Numeric]* on page 363

SORTKEY Function [String]

Generates values that can be used to sort character strings based on alternate collation rules.

Syntax

```
SORTKEY ( string-expression
[, { collation-id
| collation-name [(collation-tailoring-string)] } ]
)
```

Parameters

Parameter	Description
string-expression	The string expression must contain characters that are encoded in the character set of the database and must be STRING data type. If string-expression is NULL, the SORTKEY function returns a null value. An empty string has a different sort-order value than a null string from a database column. There is no limit on the length of the input string that the SORTKEY function can handle. The result of SORTKEY is always limited to 1024 bytes and is VARBINARY data type. If the actual results exceed 1024 bytes, the result contains only the first 1024 bytes.
collation-name	A string or character variable that specifies the name of the sort order to use. You can also specify the alias char_collation, or, equivalently, db_collation, to generate sort-keys as used by the CHAR collation in use by the database. Similarly, you can specify the alias NCHAR_COLLATION to generate sort-keys as used by the NCHAR collation in use by the database. However, SAP Sybase IQ does not support NCHAR_COLLATION for SAP Sybase IQ-specific objects. NCHAR_COLLATION is supported for SQL Anywhere objects on a SAP Sybase IQ server.

Parameter	Description
collation-id	A variable, integer constant, or string that specifies the ID number of the sort order to use. This parameter applies only to Adaptive Server Enterprise collations, which can be referred to by their corresponding collation ID.
collation-tailoring-string	(Optional) Specify collation tailoring options (*collation-tailoring-string*) for additional control over sorting and comparison of characters. These options take the form of keyword=value pairs assembled in parentheses, following the collation name. For example, `‘UCA(locale=es;case=LowerFirst;accent=respect)’` The syntax for specifying these options is identical to the **COLLATION** clause of the **CREATE DATABASE** statement. . **Note:** All of the collation tailoring options are supported for SQL Anywhere databases, when specifying the Unicode Collation Algorithm (UCA) collation. For all other collations, only case sensitivity tailoring is supported.

Returns
BINARY

Example
The following statement queries the Employees table and returns the FirstName and Surname of all employees, sorted by the sort-key values for the Surname column using the dict collation (Latin-1, English, French, German dictionary):

```
SELECT Surname, GivenName FROM Employees ORDER BY SORTKEY( Surname,
'dict' );
```

Usage
The **SORTKEY** function generates values that can be used to order results based on predefined sort order behavior. This allows you to work with character sort order behaviors that may not be available from the database collation. The returned value is a binary value that contains coded sort order information for the input string that is retained from the **SORTKEY** function.

For example, you can store the values returned by the **SORTKEY** function in a column with the source character string. The following **SELECT** statement retrieves data from table T1 in the sorted order of c1 according to the Thai dictionary:

```
SELECT rid, c1 from T1 ORDER BY SORTKEY(c1)
```

You instead store the value returned by **SORTKEY** in a column with the source character string. To retrieve the character data in the required order, the **SELECT** statement needs to include only an **ORDER BY** clause on the column that contains the results of running the **SORTKEY** function:

```
UPDATE T1 SET shadowc1=SORTKEY(c1) FROM T1;
SELECT rid, c1 FROM T1 ORDER BY shadowc1
```

The **SORTKEY** function guarantees that the values it returns for a given set of sort order criteria work for the binary comparisons that are performed on `VARBINARY` data types.

Generating sort-keys for queries can be expensive. As an alternative for frequently requested sort-keys, consider creating a computed column to hold the sort-key values, and then referencing that column in the **ORDER BY** clause of the query.

If you do not specify a collation name or collation ID, the default is Default Unicode multilingual.

Valid collations are as follows:

- To see collations that are supported by SAP Sybase IQ, listed by label, execute `iqinit -l`.
- The Adaptive Server Enterprise collations are listed in the table below.

Description	Collation Name	Collation ID
Default Unicode multilingual	default	0
CP 850 Alternative: no accent	altnoacc	39
CP 850 Alternative: lowercase first	altdict	45
CP 850 Western European: no case, preference	altnocsp	46
CP 850 Scandinavian dictionary	scandict	47
CP 850 Scandinavian: no case, preference	scannocp	48
GB Pinyin	gbpinyin	n/a
Binary sort	binary	50
Latin-1 English, French, German dictionary	dict	51
Latin-1 English, French, German no case	nocase	52
Latin-1 English, French, German no case, preference	nocasep	53
Latin-1 English, French, German no accent	noaccent	54
Latin-1 Spanish dictionary	espdict	55

Description	Collation Name	Collation ID
Latin-1 Spanish no case	espnocs	56
Latin-1 Spanish no accent	espnoac	57
ISO 8859-5 Russian dictionary	rusdict	58
ISO 8859-5 Russian no case	rusnocs	59
ISO 8859-5 Cyrillic dictionary	cyrdict	63
ISO 8859-5 Cyrillic no case	cyrnocs	64
ISO 8859-7 Greek dictionary	elldict	65
ISO 8859-2 Hungarian dictionary	hundict	69
ISO 8859-2 Hungarian no accents	hunnoac	70
ISO 8859-2 Hungarian no case	hunnocs	71
ISO 8859-5 Turkish dictionary	turdict	72
ISO 8859-5 Turkish no accents	turnoac	73
ISO 8859-5 Turkish no case	turnocs	74
CP 874 (TIS 620) Royal Thai dictionary	thaidict	1
ISO 14651 ordering standard	14651	22
Shift-JIS binary order	sjisbin	179
Unicode UTF-8 binary sort	utf8bin	24
EUC JIS binary order	eucjisbn	192
GB2312 binary order	gb2312bn	137
CP932 MS binary order	cp932bin	129
Big5 binary order	big5bin	194
EUC KSC binary order	euckscbn	161

With respect to collation tailoring, full sensitivity is generally the intent when creating sort-keys, so when you specify a non-UCA collation, the default tailoring applied is equivalent to case=Respect. For example, the following two statements are equivalent:

```
SELECT SORTKEY( 'abc', '1252LATIN1' );
SELECT SORTKEY( 'abc', '1252LATIN1(case=Respect)' );
```

When specifying a non-UCA collation, by default, collation tailorings are accent and case-sensitive. However, for non-UCA collations, you can override only the case sensitivity using a collation tailoring. For example:

```
SELECT SORTKEY( 'abc', '1252LATIN1(case=LowerFirst)' );
```

If the database was created without specifying tailoring options, the following two clauses may generate different sort orders, even if the database collation name is specified for the **SORTKEY** function:

```
ORDER BY string-expression
```

```
ORDER BY SORTKEY( string-expression, database-collation-name )
```

Different sort orders may be generated, because the default tailoring settings used for database creation and for the **SORTKEY** function are different. To get the same behavior from **SORTKEY** as for the database collation, either provide a tailoring syntax for *collation-tailoring-string* that matches the settings for the database collation, or specify `db_collation` for collation-name. For example:

```
SORTKEY( expression, 'db_collation' )
```

Note: Sort-key values created using a version of SAP Sybase IQ earlier than 15.0 do not contain the same values created using version 15.0 and later. This may be a problem for your applications if your pre-15.0 database has sort-key values stored within it, especially if sort-key value comparison is required by your application. Regenerate any sort-key values in your database that were generated using a version of SAP Sybase IQ earlier than 15.0.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

SOUNDEX Function [String]

Returns a number representing the sound of a string.

Syntax

```
SOUNDEX ( string-expression )
```

Parameters

Parameters	Description
string-expression	The string.

Returns

SMALLINT

Example

The following statement returns two numbers, representing the sound of each name. The **SOUNDEX** value for each argument is 3827.

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' ) FROM iq_dummy
```

SOUNDEX ('Smith') is equal to **SOUNDEX** ('Smythe').

Usage

The **SOUNDEX** function value for a string is based on the first letter and the next three consonants other than H, Y, and W. Doubled letters are counted as one letter. For example:

```
SOUNDEX( 'apples' ) FROM iq_dummy
```

is based on the letters A, P, L, and S.

Multibyte characters are ignored by the **SOUNDEX** function.

Although it is not perfect, **SOUNDEX** normally returns the same number for words that sound similar and that start with the same letter.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise, except that Adaptive Server Enterprise returns a CHAR(4) result and SAP Sybase IQ returns an integer.

See also

- *DIFFERENCE Function [String]* on page 215

SP_HAS_ROLE Function [System]

Returns an integer value indicating whether the invoking user has been granted a specified system privilege or user-defined role. When used for permission checking within user-defined stored procedures, **SP_HAS_ROLE** returns an error message when a user fails a permission check.

Syntax

dbo.sp_has_role(*[rolename]*, *[grant_type]*, *[throw_error]*)

Arguments

Arguments	Description
rolename	The name of a system privilege or user-defined role.
grant_type	Valid values are: ADMIN and NO ADMIN. If NULL or not specified, NO ADMIN is used by default.

Arguments	Description
throw_error	Valid values are: • **1** – display error message specified if system privilege or user-defined role is not granted to invoking user. • **0** – (default) do not display error message if specified system privilege or user-defined role is not granted to invoking user.

Result Set

Value Returned	Description
1	System privilege or user-defined role is granted to invoking user.
0 or `Permission denied: you do not have permission to execute this command/procedure.`	System privilege or user-defined role is not granted to invoking user. The error message replaces the value 0 when the `throw_error` argument is set to 1.
-1	The system privilege or user-defined role specified does not exist. No error message appears, even if the `throw_error` argument is set to 1.

Remarks

If the value of the `grant_type` argument is ADMIN, the function checks whether the invoking user has administrative privileges for the system privilege. If the value of the `grant_type` argument is NO ADMIN, the function checks whether the invoking user has privileged use of the system privilege or role.

If the `grant_type` argument is not specified, NO ADMIN is used by default and output indicates only whether the invoking user has been granted, either directly or indirectly, the specified system privilege or user-defined role.

If the `rolename` and `grant_type` arguments are both NULL and the `throw_error` argument is 1, you see an error message. You may find this useful for those stored procedures where an error message appears after certain values are read from the catalog tables rather than after the checking the presence of certain system privileges for the invoking user.

Note: A permission denied error message is returned if the arguments `rolename` and `grant_type` are set to NULL and `throw_error` is set to 1, or if all three arguments are set to NULL.

Permissions

None

Example 1

Consider the following scenario:

- `u1` has been granted the CREATE ANY PROCEDURE system privilege with the WITH NO ADMIN OPTION clause.
- `u1` has not been granted the CREATE ANY TABLE system privilege.
- `u1` has been granted the user-defined role `Role_A` with the WITH ADMIN ONLY OPTION clause.
- `Role_B` exists, but has not been granted to `u1`
- The role `Role_C` does not exist.

Based on the above scenario, this command

-
  ```
  sp_has_role 'create any procedure'
  ```

  returns the value 1, which indicates `u1` has been granted the CREATE ANY PROCEDURE system privilege.

-
  ```
  sp_has_role 'create any table'
  ```

  returns the value 0, which indicates `u1` has not been granted the CREATE ANY TABLE system privilege. No error message is returned because the `throw_error` argument is not specified.

-
  ```
  sp_has_role 'create any procedure','admin',1
  ```

  returns the `Permission denied` error message (throw_error=1). Even though `u1` has been granted the CREATE ANY PROCEDURE system privilege, `u1` has not been granted administrative rights to the system privilege.

-
  ```
  sp_has_role 'Role_A'
  ```

  returns the value 1, which indicates `u1` has been granted role `Role_A`.

-
  ```
  sp_has_role 'Role_A','admin',1
  ```

  returns the value 1, which indicates `u1` has been granted role `Role_A` with administrative rights.

-
  ```
  sp_has_role 'Role_B'
  ```

  returns the value 0, which indicates `u1` has not been granted the role `ROLE_B`. No error message is returned because the `throw_error` argument is not specified.

-
  ```
  sp_has_role 'Role_C'
  ```

  returns the value -1, which indicates the role `ROLE_C` does not exist.

-
  ```
  sp_has_role 'Role_C',NULL,1
  ```

  returns the value -1, which indicates the role `ROLE_C` does not exist.

SPACE Function [String]

Returns a specified number of spaces.

Syntax

```
SPACE ( integer-expression )
```

Parameters

Parameter	Description
integer-expression	The number of spaces to return.

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **SPACE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **SPACE** to the correct data type and size.

Example

The following statement returns a string containing 10 spaces:

```
SELECT SPACE( 10 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

SQLFLAGGER Function [Miscellaneous]

Returns the conformity of a given SQL statement to a specified standard.

Syntax

```
SQLFLAGGER ( sql-standard-string, sql-statement-string )
```

Parameters

Parameter	Description
sql-standard-string	The standard level against which to test compliance. Possible values are the same as for the SQL_FLAGGER_ERROR_LEVEL database option: • SQL:2003/Core Test for conformance to core SQL/2003 syntax. • SQL:2003/Package Test for conformance to full SQL/2003 syntax. • SQL:1999/Core Test for conformance to core SQL/1999 syntax. • SQL:1999/Package Test for conformance to full SQL/1999 syntax. • SQL:1992/Entry Test for conformance to entry-level SQL/1992 syntax. • SQL:1992/Intermediate Test for conformance to intermediate-level SQL/1992 syntax. • SQL:1992/Full Test for conformance to full-SQL/1992 syntax.
sql-statement-string	The SQL statement to check for conformance.

Returns
LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **SQLFLAGGER** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **SQLFLAGGER** to the correct data type and size.

Examples
The following statement shows an example of the message that is returned when a disallowed extension is found:

```
SELECT SQLFLAGGER( 'SQL:2003/Package', 'SELECT top 1 dummy_col FROM
sys.dummy ORDER BY dummy_col' );
```

This statement returns the message `'0AW03 Disallowed language extension detected in syntax near 'top' on line 1'`.

The following statement returns '00000' because it contains no disallowed extensions:

```
SELECT SQLFLAGGER( 'SQL:2003/Package', 'SELECT dummy_col FROM
sys.dummy' );
```

Usage

You can also use the iqsqlpp SQL Preprocessor Utility to flag any Embedded SQL that is not part of a specified set of SQL92. See *iqsqlpp SQL Preprocessor Utility* in the *Utility Guide*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

SQRT Function [Numeric]

Returns the square root of a number.

Syntax

```
SQRT ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	The number for which the square root is to be calculated.

Returns

DOUBLE

Example

The following statement returns the value 3:

```
SELECT SQRT( 9 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

SQUARE Function [Numeric]

Returns the square of the specified expression as a float.

Syntax

```
SQUARE ( numeric-expression )
```

Parameters

Parameter	Description
expression	Is a column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, the SQUARE function generates an error. The return value is of DOUBLE data type.

Usage

SQUARE function takes one argument. For example, **SQUARE** (*12.01*) returns 144.240100.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

STDDEV Function [Aggregate]

Returns the standard deviation of a set of numbers.

Syntax

```
STDDEV ( [ ALL ] expression )
```

Parameters

Parameter	Description
expression	Any numeric data type (FLOAT, REAL, or DOUBLE precision) expression.

Returns

DOUBLE

Examples

Given this data:

```
SELECT Salary FROM Employees WHERE DepartmentID = 300
```

Salary
51432.000
57090.000

Salary
42300.000
43700.00
36500.000
138948.000
31200.000
58930.00
75400.00

The following statement returns the value 32617.8446712838471:

```
SELECT STDDEV ( Salary ) FROM Employees
WHERE DepartmentID = 300
```

Given this data:

```
SELECT UnitPrice FROM Products WHERE Name = 'Tee Shirt'
```

Name	UnitPrice
Tee Shirt	9.00
Tee Shirt	14.00
Tee Shirt	14.00

The following statement returns the value 2.88675134594813049:

```
SELECT STDDEV ( UnitPrice ) FROM Products
WHERE Name = 'Tee Shirt'
```

Usage

The formula used to calculate **STDDEV** is:

$$stddev = \sqrt{variance}$$

STDDEV returns a result of data type `DOUBLE` precision floating-point. If applied to the empty set, the result is NULL, which returns NULL for a one-element input set.

STDDEV does not support the keyword DISTINCT. A syntax error is returned if you use DISTINCT with **STDDEV**.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *Windowing Aggregate Function Usage* on page 83
- *STDDEV_SAMP Function [Aggregate]* on page 350
- *VARIANCE Function [Aggregate]* on page 373

STDDEV_POP Function [Aggregate]

Computes the standard deviation of a population consisting of a numeric-expression, as a `DOUBLE`.

Syntax

```
STDDEV_POP ( [ ALL ] expression )
```

Parameters

Parameter	Description
expression	The expression (commonly a column name) whose population-based standard deviation is calculated over a set of rows.

Returns

DOUBLE

Examples

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year( ship_date ) AS Year, quarter( ship_date )
  AS Quarter, AVG( quantity ) AS Average,
  STDDEV_POP ( quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
  ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	14.2794
2000	2	27.050847	15.0270
...	...	...	...

Usage

Computes the population standard deviation of the provided *value expression* evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the population variance.

$$\sqrt{\frac{\sum (x_i - \bar{x})^2}{n}}$$

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *Windowing Aggregate Function Usage* on page 83

STDDEV_SAMP Function [Aggregate]

Computes the standard deviation of a sample consisting of a numeric-expression, as a DOUBLE.

Syntax

```
STDDEV_SAMP ( [ ALL ] expression )
```

Parameters

Parameter	Description
expression	The expression (commonly a column name) whose sample-based standard deviation is calculated over a set of rows.

Returns

DOUBLE

Examples

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year( ship_date ) AS Year, quarter( ship_date )
  AS Quarter, AVG( quantity ) AS Average,
  STDDEV_SAMP( quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
  ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	14.3218
2000	2	27.050847	15.0696
...	...	...	...

Usage

Note: STDDEV_SAMP is an alias for **STDDEV**.

Computes the sample standard deviation of the provided *value expression* evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the sample variance.

NULL returns NULL for a one-element input set.

Standard deviations are computed according to the following formula, which assumes a normal distribution:

$$\sqrt{\frac{\sum (x_i - \bar{x})^2}{(n-1)}}$$

Standards and compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *Windowing Aggregate Function Usage* on page 83
- *STDDEV Function [Aggregate]* on page 347
- *VARIANCE Function [Aggregate]* on page 373

STR Function [String]

Returns the string equivalent of a number.

Syntax

```
STR ( numeric-expression [ , length [ , decimal ] ] )
```

Parameters

Parameter	Description
numeric-expression	Any approximate numeric (FLOAT, REAL, or DOUBLE precision) expression.
length	The number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, the sign, if any, and blanks). The default is 10 and the maximum length is 255.

Parameter	Description
decimal	The number of digits to the right of the decimal point to be returned. The default is 0.

Returns
VARCHAR

Examples
The following statement returns a string of six spaces followed by 1234, for a total of ten characters:

```
SELECT STR( 1234.56 ) FROM iq_dummy
```

The following statement returns the result 1234.5:

```
SELECT STR( 1234.56, 6, 1 ) FROM iq_dummy
```

Usage
If the integer portion of the number cannot fit in the length specified, then the result is NULL. For example, the following statement returns NULL:

```
SELECT STR( 1234.56, 3 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

STR_REPLACE Function [String]

Takes three arguments as input of type `BINARY` or `STRING` and replaces any instances of the second string expression (*string_expr2*) that occur within the first string expression (*string_expr1*) with a third expression (*string_expr3*).

Syntax

```
REPLACE ( string_expr1, string_expr2, string_expr3 )
```

Parameters

Parameter	Description
string_expr1	The source string, or the string expression to be searched, expressed as `CHAR`, `VARCHAR`, `UNICHAR`, `UNIVARCHAR`, `VARBINARY`, or `BINARY` data type.

Parameter	Description
string_expr2	The pattern string, or the string expression to find within the first expression (*string_expr1*) and is expressed as `CHAR`, `VARCHAR`, `UNICHAR`, `UNIVARCHAR`, `VARBINARY`, or `BINARY` data type.
string_expr3	The replacement string expression, expressed as `CHAR`, `VARCHAR`, `UNICHAR`, `UNIVARCHAR`, `VARBINARY`, or `BINARY` data type.

Example 1

Replaces the string *def* within the string *cdefghi* with *yyy*.

```
select replace("cdefghi", "def", "yyy")
-------------
cyyyghi
(1 row(s) affected)
```

Example 2

Replaces all spaces with “toyota”

```
select str_replace ("chevy, ford, mercedes", "","toyota")
----------
chevy,toyotaford,toyotamercedes
(1 row(s) affected)
```

Example 3

Accepts NULL in the third parameter and treats it as an attempt to replace *string_expr2* with NULL, effectively turning STR_REPLACE into a “string cut” operation. Returns “abcghijklm”:

```
select str_replace("abcdefghijklm", "def", NULL)
----------
abcghijklm
(1 row affected)
```

Usage

STR_REPLACE is an alias of **REPLACE** function.

- Takes any data type as input and returns STRING or BINARY.

  For example, an empty string passed as an argument (“”) is replaced with one space (“ ”) before further evaluation occurs. This is true for both `BINARY` and `STRING` types.
- All arguments can have a combination of `BINARY` and `STRING` data types.
- The result length may vary, depending upon what is known about the argument values when the expression is compiled. If all arguments are columns or host variables assigned to constants, SAP Sybase IQ calculates the result length as:

```
result_length = ((s/p)*(r-p)+s)
WHERE
  s = length of source string
  p = length of pattern string
  r = length of replacement string
IF (r-p) <= 0, result length = s
```

- If SAP Sybase IQ cannot calculate the result length because the argument values are unknown when the expression is compiled, the result length used is 255.
- **RESULT_LEN** never exceeds 32767.

Permissions

Any user can execute **STR_REPLACE**.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.

See also

- *BIT_LENGTH Function [String]* on page 165
- *BYTE_LENGTH Function [String]* on page 166
- *CHAR_LENGTH Function [String]* on page 171
- *COL_LENGTH Function [System]* on page 175
- *DATALENGTH Function [System]* on page 189
- *LEN Function [String]* on page 260
- *LENGTH Function [String]* on page 261
- *OBJECT_NAME Function [System]* on page 291
- *OCTET_LENGTH Function [String]* on page 292

STRING Function [String]

Concatenates one or more strings into one large string.

Syntax

```
STRING ( string-expression [ , … ] )
```

Parameters

Parameter	Description
string-expression	A string. If only one argument is supplied, it is converted into a single expression. If more than one argument is supplied, they are concatenated into a single string. A NULL is treated as an empty string ('').

Returns
LONG VARCHAR

LONG NVARCHAR

LONG BINARY

Note: The result data type is a `LONG VARCHAR`. If you use **STRING** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **STRING** to the correct data type and size.

Example
The following statement returns the value testing123:

```
SELECT STRING( 'testing', NULL, 123 )
FROM iq_dummy
```

Usage
Numeric or date parameters are converted to strings before concatenation. You can also use the **STRING** function to convert any single expression to a string by supplying that expression as the only parameter.

If all parameters are NULL, **STRING** returns NULL.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

STRTOUUID Function [String]

Converts a string value to a unique identifier (UUID or GUID) value.

Syntax

STRTOUUID (*string-expression*)

Parameters

Parameter	Description
string-expression	A string in the format *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*

Returns
UNIQUEIDENTIFIER

Example

```
CREATE TABLE T (
 pk uniqueidentifier primary key,
 c1 int);
INSERT INTO T (pk, c1)
 VALUES (STRTOUUID
 ('12345678-1234-5678-9012-123456789012'), 1);
```

Usage

Converts a string in the format *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* where *x* is a hexadecimal digit, to a unique identifier value. If the string is not a valid UUID string, NULL is returned.

You can use **STRTOUUID** to insert UUID values into a SAP Sybase IQ database.

Standards and Compatibility

- SQL—Transact-SQL extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *Binary Data Types* on page 394
- *NEWID Function [Miscellaneous]* on page 280
- *UUIDTOSTR Function [String]* on page 369
- *Character Data Types* on page 385
- *Binary Data Types* on page 819

STUFF Function [String]

Deletes a number of characters from one string and replaces them with another string.

Syntax

```
STUFF ( string-expression1, start, length, string-expression2 )
```

Parameters

Parameter	Description
string-expression1	The string to be modified by the **STUFF** function.
start	The character position at which to begin deleting characters. The first character in the string is position 1.
length	The number of characters to delete.

Parameter	Description
string-expression2	The string to be inserted. To delete a portion of a string using the **STUFF** function, use a replacement string of NULL

Returns
LONG NVARCHAR

Example
The following statement returns the value “chocolate pie”:

```
SELECT STUFF( 'chocolate cake', 11, 4, 'pie' )
FROM iq_dummy
```

Usage
To delete a portion of a string using **STUFF**, use a replacement string of NULL. To insert a string using **STUFF**, use a length of zero.

The **STUFF** function will return a NULL result in the following situations:

- Any of the first three parameters is a NULL value.
- Either the start or length parameter is a negative value.
- The start parameter is greater than the length of string-expression1.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

SUBSTRING Function [String]

Returns a substring of a string.

Syntax

```
{ SUBSTRING | SUBSTR } ( string-expression, start [ , length ] )
```

Parameters

Parameter	Description
string-expression	The string from which a substring is to be returned.

Parameter	Description
start	The start position of the substring to return, in characters. A negative starting position specifies a number of characters from the end of the string instead of the beginning. The first character in the string is at position 1.
length	The length of the substring to return, in characters. A positive *length* specifies that the substring ends *length* characters to the right of the starting position, while a negative *length* specifies that the substring ends *length* characters to the left of the starting position.

Returns

LONG VARCHAR

LONG NVARCHAR

LONG BINARY

Note: The result data type is a `LONG VARCHAR`. If you use **STRING** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **STRING** to the correct data type and size.

Examples

The following statement returns “back”:

```
SELECT SUBSTRING ( 'back yard', 1 , 4 )
FROM iq_dummy
```

The following statement returns yard:

```
SELECT SUBSTR ( 'back yard', -1 , -4 )
FROM iq_dummy
```

The following statement returns 0x2233:

```
SELECT SUBSTR ( 0x112233445566, 2, 2 )
FROM iq_dummy
```

Usage

If *length* is specified, the substring is restricted to that length. If no length is specified, the remainder of the string is returned, starting at the *start* position.

Both *start* and *length* can be negative. Using appropriate combinations of negative and positive numbers, you can get a substring from either the beginning or end of the string.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

When the **ansi_substring** database option is set to ON (default), negative values are invalid.

See *SUBSTRING Function* in *Unstructured Data Analytics*.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—**SUBSTR** is not supported by Adaptive Server Enterprise. Use **SUBSTRING** instead

See also

- *CHARINDEX Function [String]* on page 173

ANSI_SUBSTRING Option [TSQL]

Controls the behavior of the SUBSTRING (SUBSTR) function when negative values are provided for the start or length parameters.

Allowed Values

ON, OFF

Default

ON

Scope

Option can be set at the database (PUBLIC) or user level. When set at the database level, the value becomes the default for any new user, but has no impact on existing users. When set at the user level, overrides the PUBLIC value for that user only. No system privilege is required to set option for self. System privilege is required to set at database level or at user level for any user other than self.

Requires the SET ANY PUBLIC OPTION system privilege to set this option. Can be set temporary for an individual connection or for the PUBLIC role. Takes effect immediately.

Description

When the `ANSI_SUBSTRING` option is set to ON, the behavior of the **SUBSTRING** function corresponds to ANSI/ISO SQL/2003 behavior. A negative or zero start offset is treated as if the string were padded on the left with noncharacters, and gives an error if a negative length is provided.

When this option is set to OFF, the behavior of the **SUBSTRING** function is the same as in earlier versions of SAP Sybase IQ: a negative start offset means an offset from the end of the string, and a negative length means the desired substring ends length characters to the left of the starting offset. Using a start offset of 0 is equivalent to a start offset of 1.

Avoid using nonpositive start offsets or negative lengths with the **SUBSTRING** function. Where possible, use the **LEFT** or **RIGHT** functions instead.

Example

These examples show the difference in the values returned by the **SUBSTRING** function based on the setting of the `ANSI_SUBSTRING` option:

```
SUBSTRING( 'abcdefgh',-2,4 );
    ansi_substring = Off ==> 'gh'
    // substring starts at second-last character
    ansi_substring = On  ==> 'gh'
    // takes the first 4 characters of
    // ???abcdefgh and discards all ?
```

```
SUBSTRING( 'abcdefgh',4,-2 );
    ansi_substring = Off ==> 'cd'
    ansi_substring = On  ==> value -2 out of range
    for destination
```

```
SUBSTRING( 'abcdefgh',0,4 );
    ansi_substring = Off ==> 'abcd'
    ansi_substring = On  ==> 'abcd'
```

SUBSTRING64 Function [String]

The **SUBSTRING64** function returns a variable-length character string of the large object column or variable parameter.

Usage

SUBSTRING64 supports searching `LONG VARCHAR` and `LONG BINARY` columns and `LONG VARCHAR` and `LONG BINARY` variables of any size of data. Currently, a SQL variable can hold up to 2GB - 1 in length.

If you are licensed to use the Unstructured Data Analytics functionality, you can use this function with large object data.

See *SUBSTRING64 Function* in *Unstructured Data Analytics*.

SUM Function [Aggregate]

Returns the total of the specified expression for each group of rows.

Syntax

```
SUM ( expression | DISTINCT column-name )
```

Parameters

Parameter	Description
expression	The object to be summed. This is commonly a column name.

Parameter	Description
DISTINCT column-name	Computes the sum of the unique values in *column-name* for each group of rows. This is of limited usefulness, but is included for completeness.

Returns

INTEGER

DOUBLE

NUMERIC

Example

The following statement returns the value 3749146.740:

```
SELECT SUM( salary )
FROM Employees
```

Usage

Rows where the specified expression is NULL are not included.

Returns NULL for a group containing no rows.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *AVG Function [Aggregate]* on page 162
- *COUNT Function [Aggregate]* on page 187
- *Windowing Aggregate Function Usage* on page 83

SUSER_ID Function [System]

Returns an integer user identification number.

Syntax

```
SUSER_ID ( [ user-name ] )
```

Parameters

Parameter	Description
user-name	The user name.

Returns
INT

Examples
The following statement returns the user identification number 1:

```
SELECT SUSER_ID ('DBA') FROM iq_dummy
```

The following statement returns the user identification number 0:

```
SELECT SUSER_ID ('SYS') FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ.

See also

- *SUSER_NAME Function [System]* on page 362
- *USER_ID Function [System]* on page 368
- *USER_NAME Function [System]* on page 368

SUSER_NAME Function [System]

Returns the user name.

Syntax

```
SUSER_NAME ( [ user-id ] )
```

Parameters

Parameter	Description
user-id	The user identification number.

Returns
LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **SUSER_NAME** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license or use **CAST** and set **SUSER_NAME** to the correct data type and size.

Examples
The following statement returns the value DBA:

```
SELECT SUSER_NAME ( 1 ) FROM iq_dummy
```

The following statement returns the value SYS:

```
SELECT SUSER_NAME ( 0 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ. In Adaptive Server Enterprise, **SUSER_NAME** returns the server user name.

See also

- *SUSER_ID Function [System]* on page 361
- *USER_ID Function [System]* on page 368
- *USER_NAME Function [System]* on page 368

TAN Function [Numeric]

Returns the tangent of a number.

Syntax

```
TAN ( numeric-expression )
```

Parameters

Parameter	Description
numeric-expression	An angle, in radians.

Returns

DOUBLE

Example

Returns the value 0.572561:

```
SELECT TAN( 0.52 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *COS Function [Numeric]* on page 183
- *ATAN2 Function [Numeric]* on page 162
- *ATAN Function [Numeric]* on page 161
- *ASIN Function [Numeric]* on page 160
- *ACOS Function [Numeric]* on page 158
- *COT Function [Numeric]* on page 184

- *SIN Function [Numeric]* on page 335

TODAY Function [Date and time]

Returns the current date. This is the historical syntax for **CURRENT DATE**.

Syntax

```
TODAY ( * )
```

Returns

DATE

Example

The following statement returns the current day according to the system clock.

```
SELECT TODAY( * ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

TRIM Function [String]

Removes leading and trailing blanks from a string.

Syntax

```
TRIM ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string to be trimmed.

Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **TRIM** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license, or use **CAST** and set **TRIM** to the correct data type and size.

Example

The following statement returns the value "chocolate" with no leading or trailing blanks.

```
SELECT TRIM( '   chocolate   ' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise. Use **LTRIM** and **RTRIM** instead

TRUNCNUM Function [Numeric]

Truncates a number at a specified number of places after the decimal point.

Syntax

```
TRUNCNUM ( numeric-expression, integer-expression )
```

Parameters

Parameter	Description
numeric-expression	The number to be truncated.
integer-expression	A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative expression specifies the number of significant digits to the left of the decimal point at which to round.

Returns

NUMERIC

Examples

The following statement returns the value 600:

```
SELECT TRUNCNUM( 655, -2 ) FROM iq_dummy
```

The following statement: returns the value 655.340:

```
SELECT TRUNCNUM( 655.348, 2 ) FROM iq_dummy
```

Usage

This function is the same as **TRUNCATE**, but does not cause keyword conflicts.

You can use combinations of **ROUND**, **FLOOR**, and **CEILING** to provide similar functionality.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *ROUND Function [Numeric]* on page 326

UCASE Function [String]

Converts all characters in a string to uppercase.

Syntax

```
UCASE ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string to be converted to uppercase.

Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **UCASE** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license, or use **CAST** and set **UCASE** to the correct data type and size.

Example

The following statement returns the value “CHOCOLATE”:

```
SELECT UCASE( 'ChocoLate' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—**UCASE** is not supported by Adaptive Server Enterprise, but **UPPER** provides the same feature in a compatible manner

See also

- *LCASE Function [String]* on page 256

- *LEFT Function [String]* on page 259
- *LOWER Function [String]* on page 269
- *REPLACE Function [String]* on page 321
- *REVERSE Function [String]* on page 324
- *RIGHT Function [String]* on page 325
- *UPPER Function [String]* on page 367

UPPER Function [String]

Converts all characters in a string to uppercase.

Syntax

```
UPPER ( string-expression )
```

Parameters

Parameter	Description
string-expression	The string to be converted to uppercase.

Returns

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **UPPER** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license, or use **CAST** and set **UPPER** to the correct data type and size.

Example

The following statement returns the value “CHOCOLATE”:

```
SELECT UPPER( 'ChocoLate' ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Compatible with Adaptive Server Enterprise.

See also

- *LCASE Function [String]* on page 256
- *LEFT Function [String]* on page 259
- *LOWER Function [String]* on page 269

- *REPLACE Function [String]* on page 321
- *REVERSE Function [String]* on page 324
- *RIGHT Function [String]* on page 325
- *UCASE Function [String]* on page 366

USER_ID Function [System]

Returns an integer user identification number.

Syntax

```
USER_ID ( [ user-name ] )
```

Parameters

Parameter	Description
user-name	The user name.

Returns

INT

Examples

The following statement returns the user identification number 1:

```
SELECT USER_ID ('DBA') FROM iq_dummy
```

The following statement returns the user identification number 0:

```
SELECT USER_ID ('SYS') FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ.

See also

- *SUSER_ID Function [System]* on page 361
- *SUSER_NAME Function [System]* on page 362
- *USER_NAME Function [System]* on page 368

USER_NAME Function [System]

Returns the user name.

Syntax

```
USER_NAME ( [ user-id ] )
```

Parameters

Parameter	Description
user-id	The user identification number.

Returns

LONG VARCHAR

Note: The result data type is a `LONG VARCHAR`. If you use **USER_NAME** in a **SELECT INTO** statement, you must have an Unstructured Data Analytics Option license, or use **CAST** and set **USER_NAME** to the correct data type and size.

Examples

The following statement returns the value "DBA":

```
SELECT USER_NAME ( 1 ) FROM iq_dummy
```

The following statement returns the value "SYS":

```
SELECT USER_NAME ( 0 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Adaptive Server Enterprise function implemented for SAP Sybase IQ. In Adaptive Server Enterprise, USER_NAME returns the user name, not the server user name.

See also

- *SUSER_ID Function [System]* on page 361
- *SUSER_NAME Function [System]* on page 362
- *USER_ID Function [System]* on page 368

UUIDTOSTR Function [String]

Converts a unique identifier value (UUID, also known as GUID) to a string value.

Syntax

```
UUIDTOSTR ( uuid-expression )
```

Parameters

Table 58. Parameters

Parameter	Description
uuid-expression	A unique identifier value.

Returns
VARCHAR

Example
To convert a unique identifier value into a readable format, execute a query similar to:

```
CREATE TABLE T3 (
pk uniqueidentifier primary key,c1 int);
INSERT INTO T3 (pk, c1)
values (0x12345678123456789012123456789012, 1)
SELECT UUIDTOSTR(pk) FROM T3
```

Usage
Converts a unique identifier to a string value in the format *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*, where x is a hexadecimal digit. If the binary value is not a valid unique identifier, NULL is returned.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also
- *Binary Data Types* on page 394
- *NEWID Function [Miscellaneous]* on page 280
- *STRTOUUID Function [String]* on page 355
- *Character Data Types* on page 385
- *Binary Data Types* on page 819

VAR_POP Function [Aggregate]

Computes the statistical variance of a population consisting of a numeric-expression, as a `DOUBLE`.

Syntax

```
VAR_POP ( [ ALL ] expression )
```

Parameters

Parameter	Description
expression	The expression (commonly a column name) whose population-based variance is calculated over a set of rows.

Returns
DOUBLE

Examples
The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year( ShipDate ) AS Year, quarter( ShipDate )
  AS Quarter, AVG( Quantity ) AS Average,
  VAR_POP( Quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
  ORDER BY Year, Quarter
```

Year	Quarter	Average	Variance
2000	1	25.775148	203.9021
2000	2	27.050847	225.8109
...	...	...	...

Usage
Computes the population variance of the provided *value expression* evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of *value expression*, from the mean of *value expression*, divided by the number of rows (remaining) in the group or partition.

Population-based variances are computed according to the following formula:

$$\frac{\sum (x_i - \bar{x})^2}{n}$$

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *Windowing Aggregate Function Usage* on page 83

VAR_SAMP Function [Aggregate]

Computes the statistical variance of a sample consisting of a numeric-expression, as a `DOUBLE`.

Syntax

```
VAR_SAMP ( [ ALL ] expression )
```

Parameters

Parameter	Description
expression	The expression (commonly a column name) whose sample-based variance is calculated over a set of rows.

Returns

DOUBLE

Examples

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year( ShipDate ) AS Year, quarter( ShipDate )
  AS Quarter, AVG( Quantity ) AS Average,
  VAR_SAMP( Quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
  ORDER BY Year, Quarter
```

Year	Quarter	Average	Variance
2000	1	25.775148	205.1158
2000	2	27.050847	227.0939
...	...	...	...

Usage

Note: VAR_SAMP is an alias of **VARIANCE**.

Computes the sample variance of *value expression* evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of *value expression*, from the mean of *value expression*, divided by one less than the number of rows (remaining) in the group or partition.

NULL returns NULL for a one-element input set in SAP Sybase IQ.

Variances are computed according to the following formula, which assumes a normal distribution:

$$\frac{\sum (x_i - \bar{x})^2}{n}$$

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *Windowing Aggregate Function Usage* on page 83

VARIANCE Function [Aggregate]

Returns the variance of a set of numbers.

Syntax

```
VARIANCE ( [ ALL ] expression )
```

Parameters

Parameter	Description
expression	Any numeric data type (`FLOAT`, `REAL`, or `DOUBLE`) expression. The expression (commonly a column name) whose sample-based variance is calculated over a set of rows.

Returns

DOUBLE

Examples

Given this data:

```
SELECT Salary FROM Employees WHERE DepartmentID = 300
```

salary
51432.000
57090.000
42300.000

salary
43700.00
36500.000
138948.000
31200.000
58930.00
75400.00

The following statement returns the value 1063923790.99999994:

```
SELECT VARIANCE ( Salary ) FROM Employees
WHERE DepartmentID = 300
```

Given this data:

```
SELECT UnitPrice FROM Products WHERE name = 'Tee Shirt'
```

UnitPrice
9.00
14.00
14.00

The following statement returns the value 8.3333333333334327:

```
SELECT VARIANCE ( UnitPrice ) FROM Products
WHERE name = 'Tee Shirt'
```

Usage

The formula used to calculate **VARIANCE** is

$$var = \frac{n\sum x^2 - \left(\sum x\right)^2}{n(n-1)}$$

VARIANCE returns a result of data type `double-precision floating-point`. If applied to the empty set, the result is NULL, which returns NULL for a one-element input set.

VARIANCE does not support the keyword DISTINCT. A syntax error is returned if DISTINCT is used with **VARIANCE**.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *Windowing Aggregate Function Usage* on page 83
- *STDDEV Function [Aggregate]* on page 347
- *STDDEV_SAMP Function [Aggregate]* on page 350

WEEKS Function [Date and Time]

Returns the number of weeks since an arbitrary starting date/time, returns the number of weeks between two specified date/times, or adds the specified integer-expression number of weeks to a date/time.

Syntax

```
WEEKS ( datetime-expression
| datetime-expression, datetime-expression
| datetime-expression, integer-expression )
```

Parameters

Parameter	Description
datetime-expression	A date and time.
integer-expression	The number of weeks to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of weeks are subtracted from the date/time value. Hours, minutes, and seconds are ignored. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a `DATETIME` data type.

Returns

Syntax 1 returns an INTEGER.

Syntax 2 returns a TIMESTAMP.

Examples

The following statement returns the value 104278:

```
SELECT WEEKS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 9, to signify the difference between the two dates:

```
SELECT WEEKS( '1999-07-13 06:07:12',
    '1999-09-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the timestamp value 1999-06-16 21:05:07.000:

```
SELECT WEEKS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5) FROM iq_dummy
```

Usage

Weeks are defined as going from Sunday to Saturday, as they do in a North American calendar. The number returned by the first syntax is often useful for determining if two dates are in the same week.

```
WEEKS ( invoice_sent ) = WEEKS ( payment_received ) FROM iq_dummy
```

In the second syntax, the value of **WEEKS** is calculated from the number of Sundays between the two dates. Hours, minutes, and seconds are ignored. This function is not affected by the `DATE_FIRST_DAY_OF_WEEK` option.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *CAST Function [Data Type Conversion]* on page 167
- *CONVERT Function [Data Type Conversion]* on page 178
- *HOURS Function [Date and Time]* on page 235
- *MINUTES Function [Date and Time]* on page 275
- *MONTHS Function [Date and Time]* on page 278
- *REPLACE Function [String]* on page 321
- *SECOND Function [Date and Time]* on page 331
- *YEAR Function [Date and Time]* on page 380
- *YEARS Function [Date and Time]* on page 381

WEIGHTED_AVG Function [Aggregate]

Calculates an arithmetically (or linearly) weighted average.

A weighted average is an average in which each quantity to be averaged is assigned a weight. Weightings determine the relative importance of each quantity that make up the average.

Syntax

WEIGHTED_AVG (*expression*)

OVER (*window-spec*)

window-spec: See the Usage section, below.

Parameters

Parameter	Description
expression	A numeric expression for which a weighted value is being computed.

Usage

Use the **WEIGHTED_AVG** function to create a weighted moving average. In a weighted moving average, weights decrease arithmetically over time. Weights decrease from the highest weight for the most recent data points, down to zero.

Figure 1: WEIGHTED_AVG Calculation

$$WMA_M = \frac{np_M + (n-1)p_{M-1} + \cdots + 2p_{M-n+2} + p_{M-n+1}}{n + (n-1) + \cdots + 2 + 1}$$

To exaggerate the weighting, you can average two or more weighted moving averages together, or use an **EXP_WEIGHTED_AVG** function instead.

You can specify elements of *window-spec* either in the function syntax (inline), or with a **WINDOW** clause in the **SELECT** statement.

window-spec:

- Must contain an ORDER BY specifier.
- Cannot contain FOLLOWING or RANGE specifiers.
- The second argument of the ROW specifier—if provided—must be CURRENT ROW.
- Cannot contain NULL values.
- Cannot contain the DISTINCT specifier.
- UNBOUNDED PRECEDING is supported, but may result in poor performance if used

Example

The following example returns a weighted average of salaries by department for employees in Florida, with the salary of recently hired employees contributing the most weight to the average:

```
SELECT DepartmentID, Surname, Salary,
WEIGHTED_AVG(Salary) OVER (PARTITION BY DepartmentID
ORDER BY YEAR(StartDate) DESC) as "W_AVG"
FROM Employees
WHERE State IN ('FL') ORDER BY DepartmentID
```

The returned result set is:

Table 59. WEIGHTED_AVG result set

DepartmentID	Surname	Salary	W_AVG
100	Lull	87,900.000	87,900.000000
100	Gowda	59,840.000	69,193.333333
200	Sterling	64,900.000	64,900.000000
200	Kelly	87,500.000	79,966.666667
300	Litton	58,930.000	58,930.000000
400	Evans	68,940.000	68,940.000000
400	Charlton	28,300.000	41,846.666667
400	Francis	53,870.000	47,858.333333

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.

See also

- *EXP_WEIGHTED_AVG Function [Aggregate]* on page 222
- *Windowing Aggregate Function Usage* on page 83

WIDTH_BUCKET Function [Numerical]

For a given expression, the **WIDTH_BUCKET** function returns the bucket number that the result of this expression will be assigned after it is evaluated.

Syntax

```
WIDTH_BUCKET ( expression, min_value, max_value, num_buckets )
```

Parameters

Parameter	Description
expression	The expression for which the histogram is being created. This expression must evaluate to a numeric or datetime value or to a value that can be implicitly converted to a numeric or datetime value. If *expr* evaluates to null, then the expression returns null.

Parameter	Description
min_value	An expression that resolves to the end points of the acceptable range for *expr*. Must also evaluate to numeric or datetime values and cannot evaluate to null.
max_value	An expression that resolves to the end points of the acceptable range for *expr*. Must also evaluate to numeric or datetime values and cannot evaluate to null.
num_buckets	Is an expression that resolves to a constant indicating the number of buckets. This expression must evaluate to a positive integer.

Examples

The following example creates a ten-bucket histogram on the `credit_limit` column for customers in Massachusetts in the sample table and returns the bucket number ("Credit Group") for each customer. Customers with credit limits greater than the maximum value are assigned to the overflow bucket, 11:

```
select EmployeeID, Surname, Salary, WIDTH_BUCKET(Salary, 29000,
60000, 4) "Wages" from Employees where State = 'FL' order by "Wages"
```

```
EMPLOYEEID   SURNAME               SALARY         Wages
----------   -------               ------         -----
888          Charlton           28300.000             0
1390         Litton             58930.000             4
207          Francis            53870.000             4
266          Gowda              59840.000             4
445          Lull               87900.000             5
1021         Sterling           64900.000             5
902          Kelly              87500.000             5
1576         Evans              68940.000             5
```

When the bounds are reversed, the buckets are open-closed intervals. For example: **WIDTH_BUCKET** (*credit_limit*, *5000*, *0*, *5*). In this example, bucket number 1 is (4000, 5000], bucket number 2 is (3000, 4000], and bucket number 5 is (0, 1000]. The overflow bucket is numbered 0 (5000, +infinity), and the underflow bucket is numbered 6 (-infinity, 0].

Usage

You can generate equiwidth histograms with the **WIDTH_BUCKET** function. Equiwidth histograms divide data sets into buckets whose interval size (highest value to lowest value) is equal. The number of rows held by each bucket will vary. A related function, **NTILE**, creates equiheight buckets.

Equiwidth histograms can be generated only for numeric, date or datetime data types; therefore, the first three parameters should be all numeric expressions or all date expressions.

Other types of expressions are not allowed. If the first parameter is NULL, the result is NULL. If the second or the third parameter is NULL, an error message is returned, as a NULL value cannot denote any end point (or any point) for a range in a date or numeric value dimension. The last parameter (number of buckets) should be a numeric expression that evaluates to a positive integer value; 0, NULL, or a negative value will result in an error.

Buckets are numbered from 0 to (n+1). Bucket 0 holds the count of values less than the minimum. Bucket(n+1) holds the count of values greater than or equal to the maximum specified value.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

YEAR Function [Date and Time]

Returns a 4-digit number corresponding to the year of the given date/time.

Syntax

```
YEAR ( datetime-expression )
```

Parameters

Parameter	Description
datetime-expression	A date and time.

Returns

SMALLINT

Example

The following statement returns the value 1998:

```
SELECT YEAR( '1998-07-13 06:07:12' ) FROM iq_dummy
```

Usage

The **YEAR** function is the same as the first syntax of the **YEARS** function.

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *CAST Function [Data Type Conversion]* on page 167
- *CONVERT Function [Data Type Conversion]* on page 178

- *HOURS Function [Date and Time]* on page 235
- *MINUTES Function [Date and Time]* on page 275
- *MONTHS Function [Date and Time]* on page 278
- *REPLACE Function [String]* on page 321
- *SECOND Function [Date and Time]* on page 331
- *WEEKS Function [Date and Time]* on page 375
- *YEARS Function [Date and Time]* on page 381
- *NTILE Function [Analytical]* on page 286

YEARS Function [Date and Time]

Returns a 4-digit number corresponding to the year of a given date/time, returns the number of years between two specified date/times, or adds the specified integer-expression number of years to a date/time.

Syntax

```
YEARS ( datetime-expression
| datetime-expression, datetime-expression
| datetime-expression, integer-expression )
```

Parameters

Parameter	Description
datetime-expressio	A date and time.
integer-expression	The number of years to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of years are subtracted from the datetime value. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a `DATETIME` data type.

Returns

Syntax 1 returns an INTEGER.

Syntax 2 returns a TIMESTAMP.

Examples

The following statement returns the value 1998:

```
SELECT YEARS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 2, to signify the difference between the two dates.

```
SELECT YEARS( '1997-07-13 06:07:12',
    '1999-09-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the YEARS(cast('1999-05-12 21:05:07' as timestamp), 5) value 2004-05-12 21:05:07.000:

```
SELECT YEARS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5) FROM iq_dummy
```

Usage

The first syntax of the **YEARS** function is the same as the **YEAR** function.

The second syntax returns the number of years from the first date to the second date, calculated from the number of first days of the year between the two dates. The number might be negative. Hours, minutes, and seconds are ignored. For example, the following statement returns 2, which is the number of first days of the year between the specified dates:

```
SELECT YEARS ( '2000-02-24', '2002-02-24' ) FROM iq_dummy
```

The next statement also returns 2, even though the difference between the specified dates is not two full calendar years. The value 2 is the number of first days of the year (in this case January 01, 2001 and January 01, 2002) between the two dates.

```
SELECT YEARS ( '2000-02-24', '2002-02-20' ) FROM iq_dummy
```

The third syntax adds an *integer-expression* number of years to the given date. If the new date is past the end of the month (such as **SELECT YEARS** (**CAST** (‘1992-02-29’ AS **TIMESTAMP**), 1)), the result is set to the last day of the month. If *integer-expression* is negative, the appropriate number of years is subtracted from the date. Hours, minutes, and seconds are ignored.

Standards and compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

See also

- *CAST Function [Data Type Conversion]* on page 167
- *CONVERT Function [Data Type Conversion]* on page 178
- *HOURS Function [Date and Time]* on page 235
- *MINUTES Function [Date and Time]* on page 275
- *MONTHS Function [Date and Time]* on page 278
- *REPLACE Function [String]* on page 321
- *SECOND Function [Date and Time]* on page 331
- *WEEKS Function [Date and Time]* on page 375
- *YEAR Function [Date and Time]* on page 380

YMD Function [Date and Time]

Returns a date value corresponding to the given year, month, and day of the month.

Syntax

```
YMD ( integer-expression1, integer-expression2, integer-
expression3 )
```

Parameters

Parameter	Description
integer-expression1	The year.
integer-expression2	The number of the month. If the month is outside the range 1–12, the year is adjusted accordingly.
integer-expression3	The day number. The day is allowed to be any integer, the date is adjusted accordingly.

Returns

DATE

Examples

The following statement returns the value 1998-06-12:

```
SELECT YMD( 1998, 06, 12 ) FROM iq_dummy
```

If the values are outside their normal range, the date adjusts accordingly. For example, the following statement returns the value 1993-03-01:

```
SELECT YMD( 1992, 15, 1 ) FROM iq_dummy
```

The following statement returns the value 1993-02-28:

```
SELECT YMD ( 1992, 15, 1-1 ) FROM iq_dummy
```

The following statement returns the value 1992-02-29:

```
SELECT YMD ( 1992, 3, 1-1 ) FROM iq_dummy
```

Standards and Compatibility

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Not supported by Adaptive Server Enterprise.

SQL Data Types

SQL data types define the type of data to be stored, such as character strings, numbers, and dates.

Character Data Types

Use character data types for storing strings of letters, numbers and symbols.

Syntax

Use this syntax for character data types.

CHAR `[ ( max-length ) ]`

CHARACTER `[ ( max-length ) ]`

CHARACTER VARYING `[ ( max-length ) ]`

VARCHAR `[ ( max-length ) ]`

UNIQUEIDENTIFIERSTR

Usage

This table describes the character data types.

Table 60. Character Data Type

Character Data Type	Description
CHAR	Character data of maximum length *max-length* bytes. If *max-length* is omitted, the default is 1. The maximum size allowed is 32KB – 1. See Notes for restrictions on `CHAR` data greater than 255 bytes. See the notes below on character data representation in the database, and on storage of long strings. All `CHAR` values are blank padded up to *max-length*, regardless of whether the `BLANK PADDING` option is specified. When multibyte character strings are held as a `CHAR` type, the maximum length is still in bytes, not characters.
CHARACTER	Same as `CHAR`.

Character Data Type	Description
CHARACTER VARYING	Same as `VARCHAR`.
LONG VARCHAR	Arbitrary length character data. The maximum size is limited by the maximum size of the database file (currently 2 gigabytes).
TEXT	This is a user-defined data type. It is implemented as a LONG VARCHAR allowing NULL.
VARCHAR	Same as CHAR, except that no blank padding is added to the storage of these strings, and VARCHAR strings can have a maximum length of (32KB – 1). See Notes for restrictions on VARCHAR data greater than 255 bytes.
UNIQUEIDENTIFIERSTR	Domain implemented as `CHAR( 36 )`. This data type is used for remote data access, when mapping Microsoft SQL Server uniqueidentifier columns.

Note: As a separately licensed option, SAP Sybase IQ supports character large object (`CLOB`) data with a length ranging from zero (0) to 512TB (terabytes) for an SAP Sybase IQ page size of 128KB or 2PB (petabytes) for an SAP Sybase IQ page size of 512KB. The maximum length is equal to 4GB multiplied by the database page size. See Unstructured Data Analytics.

See also

- *Binary Data Types* on page 394
- *NEWID Function [Miscellaneous]* on page 280
- *STRTOUUID Function [String]* on page 355
- *UUIDTOSTR Function [String]* on page 369
- *Binary Data Types* on page 819
- *Character Data Type Compatibility* on page 818

Storage Sizes

The storage size of character data, given column definition size and input data size.

Table 61. Storage Size of Character Data

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

Character Sets and Code Pages

Character data is placed in the database using the exact binary representation that is passed from the application.

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

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

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

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

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

Indexes

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

VARCHAR Data and Trailing Blanks

For a column of data type `VARCHAR`, trailing blanks within the data being inserted are handled differently depending on whether or not the data is enclosd in quotes.

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

- Enclosed in quotes
- Not enclosed in quotes
- Binary

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

- For data enclosed in quotes, trailing blanks are never trimmed.

- For data not enclosed in quotes:
  - Trailing blanks always trimmed on insert and update.
  - For a **LOAD** statement, you can use the `STRIP RTRIM/OFF` **LOAD** option to specify whether to have the trailing blanks trimmed. The `STRIP RTRIM/OFF` option applies only to variable-length non-binary data. For example, assume the following schema:

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

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

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

    LOAD TABLE t( c1 BINARY ) ..... STRIP RTRIM    // trailing
    blanks trimmed
    LOAD TABLE t( c1 BINARY ) ..... STRIP OFF     // trailing blanks
    trimmed
    ```
  - For binary data, trailing blanks are always trimmed.

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

Restriction on CHAR and VARCHAR Data Over 255 Bytes

Only the default index, **WD**, **TEXT**, and **CMP** index types are supported for `CHAR` and `VARCHAR` columns over 255 bytes.

You cannot create an **LF**, **HG**, **HNG**, **DATE**, **TIME**, or **DTTM** index for these columns.

Character Data Compatibility

Character data compatibility differences exist between SAP Sybase IQ and Adaptive Server Enterprise and SQL Anywhere.

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

Long Strings

Values up to 254 characters are stored as short strings, with a preceding length byte. Any values that are longer than 255 bytes are considered long strings. Characters after the 255th are stored separate from the row containing the long string value.

SQL Anywhere treats `CHAR`, `VARCHAR`, and `LONG VARCHAR` columns all as the same type.

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

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

Numeric Data Types

Use numeric data types for storing numerical data.

Syntax

Use this syntax for numeric data types.

```
[ UNSIGNED ] BIGINT
[ UNSIGNED ] { INT | INTEGER }
SMALLINT
TINYINT
DECIMAL [ ( precision [ , scale ] ) ]
NUMERIC [ ( precision [ , scale ] ) ]
DOUBLE
FLOAT [ ( precision ) ]
REAL
```

Usage for Numeric Data Types

Be aware of these points when using numeric data types.

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

- Do not fetch `TINYINT` columns into Embedded SQL variables defined as `CHAR` or `UNSIGNED CHAR`, since the result is an attempt to convert the value of the column to a string and then assign the first byte to the variable in the program.
- A period is the only decimal separator (decimal point); comma is not supported as a decimal separator.

Table 62. Numeric Data Type

Numeric Data Type	Description
BIGINT	A signed 64-bit integer, requiring 8 bytes of storage. You can specify integers as `UNSIGNED`. By default the data type is signed. Its range is between -9223372036854775808 and 9223372036854775807 (signed) or from 0 to 18446744073709551615 (unsigned).
INT or INTEGER	A signed 32-bit integer with a range of values between -2147483648 and 2147483647 requiring 4 bytes of storage. The `INTEGER` data type is an exact numeric data type; its accuracy is preserved after arithmetic operations. You can specify integers as UNSIGNED; by default the data type is signed. The range of values for an unsigned integer is between 0 and 4294967295.
SMALLINT	A signed 16-bit integer with a range between -32768 and 32767, requiring 2 bytes of storage. The SMALLINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.
TINYINT	An unsigned 8-bit integer with a range between 0 and 255, requiring 1 byte of storage. The `TINYINT` data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

Numeric Data Type	Description
DECIMAL	A signed decimal number with *precision* total digits and with *scale* of the digits after the decimal point. The precision can equal 1 to 126, and the scale can equal 0 up to precision value. The defaults are scale = 38 and precision = 126. Results are calculated based on the actual data type of the column to ensure accuracy, but you can set the maximum scale of the result returned to the application using the `MAX_CLIENT_NUMERIC_SCALE` option.
NUMERIC	Same as `DECIMAL`.
DOUBLE	A signed double-precision floating-point number stored in 8 bytes. The range of absolute, nonzero values is between 2.2250738585072014e-308 and 1.797693134862315708e+308. Values held as `DOUBLE` are accurate to 15 significant digits, but might be subject to rounding errors beyond the fifteenth digit. The `DOUBLE` data type is an approximate numeric data type; it is subject to rounding errors after arithmetic operations.
FLOAT	If *precision* is not supplied, the `FLOAT` data type is the same as the `REAL` data type. If *precision* supplied, then the `FLOAT` data type is the same as the `REAL` or `DOUBLE` data type, depending on the value of the precision. The cutoff between `REAL` and `DOUBLE` is platform-dependent, and it is the number of bits used in the mantissa of single-precision floating point number on the platform. When a column is created using the FLOAT data type, columns on all platforms are guaranteed to hold the values to at least the specified minimum precision. In contrast, REAL and DOUBLE do not guarantee a platform-independent minimum precision. The `FLOAT` data type is an approximate numeric data type; it is subject to rounding errors after arithmetic operations.

Numeric Data Type	Description
REAL	A signed single-precision floating-point number stored in 4 bytes. The range of absolute, nonzero values is 1.175494351e-38 to 3.402823466e+38. Values held as `REAL` are accurate to 6 significant digits, but might be subject to rounding errors beyond the sixth digit. The `REAL` data type is an approximate numeric data type; it is subject to rounding errors after arithmetic operations.

This table lists the storage required for a decimal number.

Table 63. Storage size for a decimal number

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

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

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

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

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

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

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

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

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

is described as:

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

Note: The maximum value supported in SQL Anywhere for the numeric function is 255. If the precision of the numeric function exceeds the maximum value supported in SQL Anywhere, the following error occurs: `"The result datatype for function '_funcname' exceeds the maximum supported numeric precision of 255. Please set the proper value for precision in numeric function, 'location'"`

Numeric Data Compatibility

Numeric data compatibility differences exist between SAP Sybase IQ and Adaptive Server Enterprise and SQL Anywhere.

- In embedded SQL, fetch `TINYINT` columns into 2-byte or 4-byte integer columns. Also, to send a `TINYINT` value to a database, the C variable should be an integer.
- Adaptive Server Enterprise 12.5.x versions do not support unsigned integers. (15.x versions support unsigned integer datatypes.) You can map SAP Sybase IQ unsigned integers to Adaptive Server Enterprise signed integers or numeric data, and the data are converted implicitly.
  - Map SAP Sybase IQ `UNSIGNED SMALLINT` data to `ASE INT`
  - If you have negative values, map SAP Sybase IQ `UNSIGNED BIGINT` to `ASE NUMERIC` (*precision*, *scale*)

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

Database	Default Precision	Default Scale
SAP Sybase IQ	126	38
Adaptive Server Enterprise	18	0
SQL Anywhere	30	6

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

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

Indexes

This section describes the relationship between index types and numeric data types.

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

Binary Data Types

Use binary data types for storing raw binary data, such as pictures, in a hexadecimal-like notation, up to a length of (32K – 1) bytes.

Syntax

```
BINARY [ ( length ) ]
```

```
VARBINARY [ ( max-length ) ]
```

```
UNIQUEIDENTIFIER
```

See also

- *NEWID Function [Miscellaneous]* on page 280
- *STRTOUUID Function [String]* on page 355
- *UUIDTOSTR Function [String]* on page 369
- *Character Data Types* on page 385
- *Binary Data Types* on page 819

Usage for Binary Data Types

Binary data begins with the characters “0x” or “0X” and can include any combination of digits and the uppercase and lowercase letters A through F.

You can specify the column length in bytes, or use the default length of 1 byte. Each byte stores 2 hexadecimal digits. Even though the default length is 1 byte, it is recommended that you always specify an even number of characters for `BINARY` and `VARBINARY` column length. If you enter a value longer than the specified column length, SAP Sybase IQ truncates the entry to the specified length without warning or error.

Binary Data Type	Description
BINARY	Binary data of length *length* bytes. If *length* is omitted, the default is 1 byte. The maximum size allowed is 32767 bytes. Use the fixed-length binary type `BINARY` for data in which all entries are expected to be approximately equal in length. Because entries in `BINARY` columns are zero-padded to the column length *length*, they might require more storage space than entries in `VARBINARY` columns.
VARBINARY	Binary data up to a length of *max-length* bytes. If *max-length* is omitted, the default is 1 byte. The maximum size allowed is (32K – 1) bytes. Use the variable-length binary type `VARBINARY` for data that is expected to vary greatly in length.
UNIQUEIDENTIFIER	The `UNIQUEIDENTIFIER` data type is used for storage of UUID (also known as GUID) values.

Treatment of Trailing Zeros

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

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

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

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

bnot	bnull	vbnot	vbnull
0x3030323731	0x3030323731	0x3030323731	0x3030323731

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

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

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

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

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

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

col_bin
0x3030323731303030

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

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

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

or :

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

ASCII Data From a Flat File

Any ASCII data loaded from a flat file into a binary type column (`BINARY` or `VARBINARY`) is stored as nibbles.

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

Storage Size

Familiarize yourself with the storage size of binary data.

Table 64. Storage size of binary data

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

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

For platform-independent conversions between hexadecimal strings and integers, use the **INTTOHEX** and **HEXTOINT** functions rather than the platform-specific **CONVERT** function.

See also

- *Data Type Conversion Functions* on page 88
- *Data Type Conversions* on page 409

String Operators

The concatenation string operators || and + both support binary type data.

Explicit conversion of binary operands to character data types is not necessary with the || operator. Explicit and implicit data conversion produce different results, however.

Restrictions on BINARY and VARBINARY Data

Restrictions apply to columns containing `BINARY` and `VARBINARY` data.

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

Binary Data Compatibility

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

Table 65. Treatment of trailing zeros

Data type	SAP Sybase IQ	SQL Anywhere	Adaptive Server Enterprise
`BINARY NOT NULL`	Padded	Not padded	Padded
`BINARY NULL`	Padded	Not padded	Not padded
`VARBINARY NOT NULL`	Truncated, not padded	Truncated, not padded	Truncated, not padded
`VARBINARY NULL`	Truncated, not padded	Truncated, not padded	Truncated, not padded

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

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

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

UNIQUEIDENTIFIER

The `UNIQUEIDENTIFIER` data type is used for storage of UUID (also known as GUID) values.

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

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

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

If you execute the following statement,

```
SELECT NEWID()
```

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

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

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

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

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

Standards and compatibility for UNIQUEIDENTIFIER

These standards and compatibilities apply to `UNIQUEIDENTIFIER` values.

- SQL—Vendor extension to ISO/ANSI SQL grammar.
- Sybase—Supported by SQL Anywhere. Not supported by Adaptive Server Enterprise.
- Backwards compatibility—In databases created before SAP Sybase IQ version 12.7, the **STRTOUUID**, **UUIDTOSTR**, and **NEWID** functions were supported through CIS functional compensation. In versions 15.3 and later, the **STRTOUUID**, **UUIDTOSTR**, and **NEWID** functions are native SAP Sybase IQ functions.

Binary Large Object Data

As a separately licensed option, SAP Sybase IQ supports binary large object (BLOB) data with a length ranging from zero (0) to 512TB (terabytes) for a page size of 128KB or 2PB (petabytes) for a page size of 512KB.

The maximum length is equal to 4GB multiplied by the database page size. See *Unstructured Data Analytics*

Bit Data Type

Use the `BIT` data type for storing Boolean values.

Data type	Values	Supported by
BIT	0 or 1	SQL Anywhere and Adaptive Server Enterprise

Usage

`BIT` stores only the values 0 or 1.

Inserting any nonzero value into a `BIT` column stores a 1 in the column. Inserting any zero value into a `BIT` column stores a 0.

Only the default index type is supported for `BIT` data.

Bit Data Compatibility

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

Date and Time Data Types

Use date and time data types for storing dates and times.

Syntax

Use this syntax for date and time data types.

```
DATE
```

```
DATETIME
```

```
SMALLDATETIME
```

```
TIME
```

```
TIMESTAMP
```

See also

- *TIMESTAMP Special Value* on page 66
- *CURRENT TIMESTAMP Special Value* on page 65
- *CURRENT TIME Special Value* on page 64
- *CURRENT DATE Special Value* on page 64
- *Retrieve Dates and Times* on page 403
- *BIGTIME and BIGDATETIME Support* on page 821

Usage for Date and Time Data Types

Familiarize yourself with these usage considerations before using date and time data types.

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

Date and Time Data Type	Description
DATETIME	A domain, implemented as `TIMESTAMP`. `DATETIME` is provided primarily for compatibility with Adaptive Server Enterprise.
SMALLDATETIME	A domain, implemented as `TIMESTAMP`. `SMALLDATETIME` is provided primarily for compatibility with Adaptive Server Enterprise.
TIME	Time of day, containing hour, minute, second, and fraction of a second. The fraction is stored to 6 decimal places. A `TIME` value requires 8 bytes of storage. (ODBC standards restrict `TIME` data type to an accuracy of seconds. For this reason, do not use `TIME` data types in **WHERE** clause comparisons that rely on a higher accuracy than seconds.)
TIMESTAMP	Point in time, containing year, month, day, hour, minute, second, and fraction of a second. The fraction is stored to 6 decimal places. The day must be a nonzero value. A `TIMESTAMP` value requires 8 bytes of storage.

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

```
create table mydates (id int, descript char(20),
  datetime_null datetime, timestamp_null timestamp);
```

```
insert into mydates values (1, 'example', '7911-12-30
  23:59:59','7911-12-30 06:03:44');
commit;
```

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

```
select * from mydates;
```

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

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

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

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

See also

- *Compatibility of String to Datetime Conversions* on page 409

Index Types Supported

These index types are supported by date and time data.

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

Send Dates and Times

You send dates and times to the database in these ways.

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

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

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

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

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

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

Retrieve Dates and Times

You retrieve dates and times from the database in these ways.

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

Usage

When a date or time is retrieved as a string, it is retrieved in the format specified by the database options `DATE_FORMAT`, `TIME_FORMAT` and `TIMESTAMP_FORMAT`.

The following operators are allowed on dates:

Table 66. Operators

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

See also

- *TIMESTAMP Special Value* on page 66
- *CURRENT TIMESTAMP Special Value* on page 65
- *CURRENT TIME Special Value* on page 64
- *CURRENT DATE Special Value* on page 64
- *Date and Time Data Types* on page 400

Compare Dates and Times

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

Usage

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

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

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

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

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

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

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

Unambiguous Dates and Times

Using the unambiguous date format prevents misinterpretation of dates according to the user's `DATE_ORDER` setting.

Usage

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

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

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

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

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

In other contexts, you can use a more flexible date format. SAP Sybase IQ can interpret a wide range of strings as formats. The interpretation depends on the setting of the `DATE_ORDER` database option. The `DATE_ORDER` database option can have the value ‘MDY’, ‘YMD’, or ‘DMY’. For example, to set the `DATE_ORDER` option to ‘DMY’ enter:

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

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

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

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

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

Specify the year using the 4-digit format.

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

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

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

- year—1900
- month—No default
- day—1 (useful for month fields; for example, ‘May 1999’ is the date ‘1999-05-01 00:00’)
- hour, minute, second, fraction—0

Domains

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

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

Simple Domains

You create domains using the `CREATE DOMAIN` statement.

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

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

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

Requires CREATE DATATYPE system privlege. Once a data type is created, the user ID that executed the **CREATE DOMAIN** statement is the owner of that data type. Any user can use the data type, and unlike other database objects, the owner name is never used to prefix the data type name.

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

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

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

```
DROP DOMAIN street_address
```

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

Constraints and Defaults with User-Defined Data Types

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

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

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

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

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

CREATE DOMAIN Statement

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

- **Example 1 –** Create 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.

Note: 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.

The **CREATE DOMAIN** statement lets you incorporate a rule, called a CHECK condition, into the definition of a user-defined data type.

SAP 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 the CREATE DATATYPE or CREATE ANY OBJECT system privilege in order to drop a user-defined data type.

Side effects:

- Automatic commit

Standards

- SQL—ISO/ANSI SQL compliant.
- 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

Requires one of:

- CREATE DATATYPE system privilege.
- CREATE ANY OBJECT system privilege.

Domain Compatibility

Domain compatibility differences exist between SAP Sybase IQ and Adaptive Server Enterprise and SQL Anywhere.

- Named constraints and defaults—In SAP Sybase IQ, user-defined data types are created with a base data type, and optionally, a NULL or NOT NULL condition. Named constraints and named defaults are not supported.
- Creating data types—In SAP Sybase IQ, you can use either the `sp_addtype` system procedure or the **CREATE DOMAIN** statement to add a domain. In Adaptive Server Enterprise, you must use `sp_addtype`. The owner of the `sp_addtype` and other stored procedures inherited from SQL Anywhere is dbo. The creator for any object created using SQL Anywhere stored procedure is also dbo, and thus a user without the ALTER DATATYPE or ALTER ANY OBJECT system privilege and DROP DATATYPE or DROP ANY OBJECT system privileges may not alter or drop domains created using `sp_addtype`. You must have these system privileges for altering and dropping domains owned by dbo.

Data Type Conversions

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

Usage

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

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

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

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

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

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

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

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

See also

- *Data Type Conversion Functions* on page 88
- *Storage Size* on page 397

Compatibility of String to Datetime Conversions

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

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

Usage

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

- Adaptive Server Enterprise converts the values below as shown.

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

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

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

See also

- *Usage for Date and Time Data Types* on page 400

Compatibility of Exported Dates

For dates in the first 9 days of a month and hours less than 10, Adaptive Server Enterprise supports a blank for the first digit; SAP Sybase IQ supports a zero or a blank.

For details on supported and unsupported Adaptive Server Enterprise data types, see *Administration: Load Management*.

Conversion of BIT to BINARY Data Type

SAP Sybase IQ supports `BIT` to `BINARY` and `BIT` to `VARBINARY` implicit and explicit conversion and is compatible with Adaptive Server Enterprise support of these conversions.

SAP Sybase IQ implicitly converts `BIT` to `BINARY` and `BIT` to `VARBINARY` data types for comparison operators, arithmetic operations, and **INSERT** and **UPDATE** statements.

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

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

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

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

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

Given the following tables and data:

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

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

Implicit conversion of `BIT` to `BINARY`:

```
INSERT tbin SELECT c2 FROM tbar

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

Implicit conversion of `BIT` to `VARBINARY`:

```
INSERT tvarbin SELECT c2 FROM tbar

c2
---
0x01
0x00
```

Explicit conversion of `BIT` to `BINARY`:

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

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

Explicit conversion of `BIT` to `VARBINARY`:

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

c2
---
0x01
0x00
```

Conversion Between BIT and CHAR/VARCHAR Data Types

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

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

Given the following tables and data:

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

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

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

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

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

Implicit conversion of `VARCHAR` to `BIT`:

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

c2
--
0
1
```

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

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

c1,char_length(tchar.c1)
```

```
------------------------
‘1’,9
‘0’,9
```

Explicit conversion of `CHAR` to `BIT`:

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

c2
--
0
1
```

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

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

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

Explicit conversion of `VARCHAR` to `BIT`:

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

c2
--
0
1
```

Differences from Other SQL Dialects

SAP Sybase IQ conforms to the ANSI SQL89 standard, but has many additional features that are defined in the IBM DB2 and SAA specifications, as well as in the ANSI SQL92 standard.

Certain SAP Sybase IQ features are not found in many other SQL implementations.

Dates

SAP Sybase IQ has date, time, and timestamp types that include year, month, day, hour, minutes, seconds, and fraction of a second. For insertions or updates to date fields, or comparisons with date fields, a free-format date is supported.

In addition, the following operations are allowed on dates:

Table 67. Date Operations

Date Operation	Description
date + integer	Add the specified number of days to a date.
date - integer	Subtract the specified number of days from a date.
date - date	Compute the number of days between two dates.
date + time	Make a timestamp out of a date and time.

Also, many functions are provided for manipulating dates and times.

Integrity

SAP Sybase IQ supports both entity and referential integrity.

This has been implemented via the following two extensions to the **CREATE TABLE** and **ALTER TABLE** statements.

```
PRIMARY KEY ( column-name, ... )
[NOT NULL] FOREIGN KEY [role-name]
                 [(column-name, ...)]
             REFERENCES table-name [(column-name, ...)]
                 [ CHECK ON COMMIT ]
```

The `PRIMARY KEY` clause declares the primary key for the relation. SAP Sybase IQ will then enforce the uniqueness of the primary key, and ensure that no column in the primary key contains the NULL value.

The `FOREIGN KEY` clause defines a relationship between this table and another table. This relationship is represented by a column (or columns) in this table which must contain values in the primary key of another table. The system then ensures referential integrity for these columns; whenever these columns are modified or a row is inserted into this table, these columns are checked to ensure that either one or more is NULL or the values match the corresponding columns for some row in the primary key of the other table.

Joins

SAP Sybase IQ allows **automatic joins** between tables.

In addition to the **NATURAL** and **OUTER** join operators supported in other implementations, SAP Sybase IQ allows **KEY** joins between tables based on foreign-key relationships. This reduces the complexity of the **WHERE** clause when performing joins.

Updates

SAP Sybase IQ allows more than one table to be referenced by **UPDATE**.

Views defined on more than one table can also be updated. Many SQL implementations do not allow updates on joined tables.

Altering Tables

ALTER TABLE has been extended.

In addition to changes for entity and referential integrity, the following types of alterations are allowed:

```
DELETE column
RENAME new-table-name
RENAME old-column TO new-column
```

You can use **MODIFY** to change the maximum length of a character column, as well as converting from one data type to another.

Subqueries Not Always Allowed

Unlike SQL Anywhere, SAP Sybase IQ does not allow subqueries to appear wherever expressions are allowed.

SAP Sybase IQ supports subqueries only as allowed in the SQL-1989 grammar, plus in the **SELECT** list of the top level query block or in the **SET** clause of an **UPDATE** statement. SAP Sybase IQ does not support SQL Anywhere extensions.

Many SQL implementations allow subqueries only on the right side of a comparison operator. For example, the following command is valid in SAP Sybase IQ but not valid in most other SQL implementations.

```
SELECT        SurName,
            BirthDate,
            (    SELECT DepartmentName
                FROM Departments
                WHERE DepartmentID = Employees.EmployeeID
                AND DepartmentID = 200 )
FROM Employees
```

Additional Functions

SAP Sybase IQ supports several functions not in the ANSI SQL definition.

See also

- *SQL Functions* on page 79

Cursors

When using Embedded SQL, cursor positions can be moved arbitrarily on the FETCH statement. Cursors can be moved forward or backward relative to the current position or a given number of records from the beginning or end of the cursor.

Physical Limitations

Limitations exist on the size of objects and the number of objects in SAP Sybase IQ databases. In most cases, computer memory and disk drive are more limiting factors.

For limitations that apply to only one platform, see the platform-specific documentation.

Table 68. Database Object Size and Number Limitations

Item	Limitation
Catalog file size	Maximum is 1TB for all platforms. Windows systems with NTFS support the 1TB maximum.
Database name size	250 bytes
Database size	Maximum database size approximates the number of files times the file size on a particular platform, depending on the maximum disk configuration. Refer to your operating system documentation for kernel parameters that affect the maximum number of files.
Dbfile size	Determined by the operating system file size
Dbspace size	Raw device: Maximum size is 4T. File system device: Maximum size is 4TB Operating system file: Maximum size supported by the operating system Creating dbspaces on NAS (Network Attached Storage) devices is not recommended.
Field size	255 bytes for `BINARY`, 32,767 bytes for `VARBINARY` 32,767 for `CHAR`, `VARCHAR` Up to 512 TB for 128 KB pages or 1 PB for 512 KB pages for `LONG BINARY`, `LONG VARCHAR`
IQ page size	Must be between 64KB and 512KB
Maximum key size	255 bytes for single-column index. 5300 bytes for multicolumn index
Maximum length of string literal	32KB

Item	Limitation
Maximum length of SQL statement	The maximum length of SQL statements is limited to the amount of memory available for the IQ catalog, and to the size of the catalog stack If your SQL statements are long, increase the catalog stack size using **-gss**, and increase catalog memory cache size using **-c** or a combination of **-ch** and **-cl** When printing the SQL statement in error messages, the text is limited to the IQ catalog page size. To print long commands, you can start the server with an increased **-gp** setting, although in general Sybase recommends that you use the default of **-gp** 4096.
Maximum length of variable-length FILLER column	512 bytes
Maximum number of dbfiles	The total number of files that can be opened depends on the number of unique file descriptors that an operating system can support
Maximum number of users (connected and concurrent)	1000 on 64-bit platforms AIX, HP, Linux, and Sun Solaris 200 on 64-bit platforms on Windows
Maximum size of temp extract file	Set by `TEMP_EXTRACT_SIZEn` option. Platform limits are: AIX & HP-UX: 0 – 64GB Sun Solaris: 0 – 512GB Windows: 0 – 128GB Linux: 0 – 512GB
Table_name size	128 bytes
Number of columns per table	SAP Sybase IQ supports up to 45,000 columns in a table. You may see performance degradation if you have more than 10,000 columns in a table. The limit on the number of columns per table that allow NULLs is approximately 8*(database-page-size - 30).
Column_name size	128 bytes
Number of events per database	2^31 – 1 = 2 147 483 647
Number of files per database	Operating system limit that user can adjust; for example, using `NOFILE`. Typically, 2047 files per database
Number of indexes	2^32 (~4,000,000) per table

Item	Limitation
Index_name size	128 bytes
Number of rows per table	Limited by table size, upper limit 2^48 -1
Number of stored procedures per database	2^32 – 1 = 4 294 967 295
Number of tables or views in a single **FROM** clause	16 – 64, depending on the query, with join optimizer turned on
Number of tables or views referenced per query	512
Number of tables per database	4,293,918,719
Number of tables referenced per transaction	No limit.
Number of UNION branches per query	512. If each branch has multiple tables in the **FROM** clause, the limit on tables per query reduces the number of **UNION** branches allowed.
Number of values in an IN list	250,000
Row size	Sybase recommends a limit of half the page size
Table size	Limited by database size

See also

- *String Functions* on page 95

System Procedures

Use the system-supplied stored procedures in SAP Sybase IQ databases to retrieve system information.

SAP Sybase IQ includes the following kinds of system procedures:

- System functions that are implemented as stored procedures.
- Catalog stored procedures, for displaying system information in tabular form.
- Multiplex stored procedures, which include both of the above types of procedures, for multiplex server operations. See *System Procedures* in *Administration: Multiplex*.
- Transact-SQL system and catalog procedures.

System stored procedures related specifically to Large Object data, including **sp_iqsetcompression** and **sp_iqshowcompression**, are described in *Stored Procedure Support* in *Unstructured Data Analytics*.

Managing Privileged System Procedure Execution

There are two security models under which privileged system procedures can run. Each model grants the ability to run the system procedure differently.

Note: The following information applies to SAP Sybase IQ privileged system procedures only, not user-defined stored procedures.

The first model, called the SYSTEM PROCEDURE DEFINER model, runs a privileged system procedure with the privileges of its owner, typically dbo. The second model, called the SYSTEM PROCEDURE INVOKER model, runs a privileged system procedure with the privileges of the person executing it.

To run a privileged system procedure using the SYSTEM PROCEDURE DEFINER model, grant explicit EXECUTE privilege on the procedure. Any system privileges required to run any underlying authorized tasks of the system procedure are automatically inherited from the owner (definer of the system procedure).

For privileged system procedures using the SYSTEM PROCEDURE INVOKER model, the EXECUTE privilege is granted to the PUBLIC role, and since by default every user is a member of the PUBLIC role, every user automatically inherits the EXECUTE privilege. However, since the PUBLIC role is not the owner of the system procedures, and is not granted any system privileges, system privileges required to run any underlying authorized tasks must be granted directly or indirectly to the user.

By default, a new 16.0 database runs all privileged system procedures using the SYSTEM PROCEDURE INVOKER model. By default, a pre-16.0 upgraded database runs privileged system procedures using a combination of both the SYSTEM PROCEDURE DEFINER and

SYSTEM PROCEDURE INVOKER models. In the combined model, all pre-16.0 privileged system procedures run using the SYSTEM PROCEDURE DEFINER model, and any privileged system procedures introduced with 16.0 (or any future release) run using the SYSTEM PROCEDURE INVOKER model. The default security model can be overridden during database creation or upgrade, or changed any time thereafter. However, this is not recommended as it may result in loss of functionality on custom stored procedures and applications.

Grant the Ability to Run a Privileged System Procedure

The process by which you grant the ability to run a privileged system procedure is dependant on the security model under which it runs.

For a privileged system procedure using the SYSTEM PROCEDURE DEFINER model, grant EXECUTE privilege on the system procedure to the user:

```
GRANT EXECUTE ON sys_procedure_name
    TO grantee [,...]
```

For a privileged system procedure using the SYSTEM PROCEDURE INVOKER model, grant the underlying system privileges required by the system procedure to the user. Use **sp_proc_priv()** to identify the system privileges required to run a system procedure.

```
GRANT system_privilege_name
    TO grantee [,...]
```

Revoke the Ability to Run a Privileged System Procedure

The process by which you revoke the ability to run a privileged system procedure is dependant on the security model under which it runs.

For a privileged system procedure using the SYSTEM PROCEDURE DEFINER model, revoke the EXECUTE privilege on the system procedure from the user:

```
REVOKE EXECUTE ON sys_procedure_name
    FROM grantee [,...]
```

For a privileged system procedure using the SYSTEM PROCEDURE INVOKER model, revoke the underlying system privileges required by the system procedure from the user:

```
REVOKE system_privilege_name
    FROM grantee [,...]
```

Determine Security Model Used by a Database

By default, a new 16.0 database runs privileged system procedures using the SYSTEM PROCEDURE INVOKER model only, while a pre-16.0 upgraded database runs privileged system procedures using a combination of both the SYSTEM PROCEDURE DEFINER and SYSTEM PROCEDURE INVOKER models.

However, it is possible to override the defaults during database creation or upgrade. To verify the security model of a database, execute:

```
select IF ((HEXTOINT(substring(db_property('Capabilities'),
1,length(db_property('Capabilities'))-20)) & 8) = 8)
THEN 1
ELSE 0
END IF
```

1 indicates the database is using the SYSTEM PROCEDURE INVOKER model. 0 indicates that the database is using the combined model.

Remember, in the combined model, only pre-16.0 privileged system procedures run using the SYSTEM PROCEDURE DEFINER. Refer to the pre-16.0 privileged system procedures list to identify these system procedures.

A new or upgraded 16.0 database cannot be configured to run all system procedures using the SYSTEM PROCEDURE DEFINER model.

Pre-16.0 Privileged System Procedures

A list of pre-16.0 privileged system procedures.

Pre-16.0 privileged system procedures that use the combined security model

For these privileged system procedures, if the database is configured to run using SYSTEM PROCEDURE DEFINER, you only need EXECUTE privilege on the procedure to run it. If the database is configured to run using SYSTEM PROCEDURE INVOKER, you need the individual privileges that each procedure requires to run successfully. Refer to the documentation for each procedure's required system privileges.

- sa_audit_string
- sa_checkpoint_execute
- sa_clean_database
- sa_column_stats
- sa_conn_activity
- sa_conn_compression_info
- sa_conn_info
- sa_conn_list
- sa_conn_options
- sa_conn_properties
- sa_db_info
- sa_db_list
- sa_db_properties
- sa_disable_auditing_type
- sa_disk_free_space
- sa_enable_auditing_type
- sa_external_library_unload
- sa_flush_cache
- sa_flush_statistics
- sa_get_histogram
- sa_get_request_profile
- sa_get_request_times
- sa_get_table_definition
- sa_get_user_status
- sa_index_density
- sa_index_levels
- sa_install_feature
- sa_java_loaded_classes
- sa_list_external_library
- sa_load_cost_model
- sa_materialized_view_can_be_immediate
- sa_procedure_profile
- sa_procedure_profile_summary
- sa_recompile_views
- sa_refresh_materialized_views
- sa_refresh_text_indexes
- sa_remove_index_consultant_analysis
- sa_text_index_vocab
- sa_text_index_vocab_nchar
- sa_unload_cost_model
- sa_user_defined_counter_add
- sa_user_defined_counter_set
- sa_validate
- sp_iq_reset_identity
- sp_iqaddlogin
- sp_iqbackupdetails
- sp_iqbackupsummary
- sp_iqcardinality_analysis
- sp_iqcheckdb
- sp_iqcheckoptions
- sp_iqclient_lookup
- sp_iqcolumn
- sp_iqcolumnuse
- sp_iqconnection
- sp_iqconstraint
- sp_iqcontext
- sp_iqcopyloginpolicy
- sp_iqcursorinfo
- sp_iqdatatype
- sp_iqdbsize
- sp_iqdbspace
- sp_iqdbspaceinfo
- sp_iqdbspaceobjectinfo
- sp_iqdbstatistics
- sp_iqdroplogin
- sp_iqemptyfile
- sp_iqestdbspaces
- sp_iqestspace
- sp_iqevent
- sp_iqfile
- sp_iqhelp
- sp_iqmodifylogin
- sp_iqmpxcheckdqpconfig
- sp_iqmpxdumptlvlog
- sp_iqmpxfilestatus
- sp_iqmpxincconnpoolinfo
- sp_iqmpxincheartbeatinfo
- sp_iqmpxinfo
- sp_iqmpxversioninfo
- sp_iqobjectinfo
- sp_iqpkeys
- sp_iqprocedure
- sp_iqprocparm
- sp_iqrebuildindex
- sp_iqrename
- sp_iqrestoreaction
- sp_iqrowdensity
- sp_iqsetcompression
- sp_iqsharedtempdistrib
- sp_iqshowcompression
- sp_iqshowpsexe
- sp_iqspaceinfo
- sp_iqspaceused
- sp_iqstatistics
- sp_iqstatus
- sp_iqsysmon
- sp_iqtable
- sp_iqtablesize
- sp_iqtableuse
- sp_iqtransaction
- sp_iqunusedcolumn
- sp_iqunusedindex
- sp_iqunusedtable
- sp_iqversionuse
- sp_iqview
- sp_iqwho
- sp_iqworkmon
- st_geometry_load_shapefile
- xp_cmdshell

• sa_reset_identity • sa_save_trace_data • sa_send_udp • sa_server_option • sa_table_fragmentation • sa_table_page_usage • sa_table_stats • sa_text_index_stats	• sp_iqindex • sp_iqindex_alt • sp_iqindexadvice • sp_iqindexfragmentation • sp_iqindexinfo • sp_iqindexmetadata • sp_iqindexsize • sp_iqindexuse • sp_iqlmconfig • sp_iqlocks • sp_iqmodifyadmin	• xp_read_file • xp_sendmail • xp_startmail • xp_startsmtp • xp_stopmail • xp_stopsmtp • xp_write_file

Pre-16.0 privileged system procedures that run with invoker privileges regardless of the security model

These pre-16.0 privileged system procedures run with the privileges of the user running the procedure, not the owner of the procedure, regardless of the security model setting. This means that in addition to requiring EXECUTE privilege on the system procedure, the user must be granted additional system privileges required by the system procedure. Refer to the documentation for the required system privileges.

- sa_describe_shapefile
- sa_get_user_status
- sa_locks
- sa_performance_diagnostics
- sa_report_deadlocks
- sa_text_index_stats

Syntax Rules for Stored Procedures

Use of parentheses and quotes in stored procedure calls varies, depending on whether you enter the procedure name directly, as you can in Interactive SQL, or invoke it with a **CALL** statement.

Some variations are permitted because the product supports both SAP Sybase IQ SQL and Transact-SQL syntax. If you need Transact-SQL compatibility, be sure to use Transact-SQL syntax.

Table 69. Stored Procedure Syntax Variations

Syntax	Syntax Type	Explanation
`procedure_name ('param')`	SAP Sybase IQ	Quotes are required if you enclose parameters in parentheses.
`procedure_name 'param'`	SAP Sybase IQ	Parentheses are optional if you enclose parameters in quotes.
`procedure_name param`	Transact-SQL	If you omit quotes around parameters, you must also omit parentheses. **Note:** Quotes are always required around parameters when the owner is specified. For example, assuming the owner is *dba*, `sp_iqtablesize 'dba.emp1'` requires quotes around the parameters. `sp_iqtablesize emp1` does not.
`procedure_name`	SAP Sybase IQ or Transact-SQL	Use this syntax if you run a procedure with no parameters directly in Interactive SQL, and the procedure has no parameters.
`call procedure_name (param='value')`	SAP Sybase IQ	Use this syntax to call a procedure that passes a parameter value.

When you use Transact-SQL stored procedures, you must use the Transact-SQL syntax.

Understanding Statistics Reported by Stored Procedures

Many stored procedures report information on the state of SAP Sybase IQ at the time the procedure executes.

This means that you get a snapshot view. For example, a report column that lists space in use by a connection shows only the space in use at the instant the procedure executes, not the maximum space used by that connection.

To monitor SAP Sybase IQ usage over an extended period, use the SAP Sybase IQ monitor, which collects and reports statistics from the time you start the monitor until you stop it, at an interval you specify.

SQL Anywhere Supported Procedures

SAP Sybase IQ supports SQL Anywhere system procedures.

Tip: SQL Anywhere stored procedures do not contain `iq` in the procedure name.

The **sa_get_table_definition** procedure is only supported for SQL Anywhere tables. If run against an SAP Sybase IQ table, the procedure returns the error `not implemented for IQ tables`.

Alphabetical List of System Stored Procedures

System stored procedures carry out System Administrator tasks in the IQ main store.

System stored procedures are owned by the user ID `dbo`.

Note: By default, the maximum length of column values displayed by Interactive SQL Classic is 30 characters. This might be inadequate for displaying output of stored procedures such as **sp_iqstatus**. To avoid truncated output, increase the length by selecting Command > Options from the Interactive SQL menu select and enter a higher value for Limit Display Columns, Limit Output Columns, or both.

sp_iqaddlogin Procedure

Adds a new SAP Sybase IQ user account to the specified login policy.

Syntax1

```
call sp_iqaddlogin (‘username_in’, ‘pwd’,
[ ’password_expiry_on_next_login ’] [ , ’policy_name ’] )
```

Syntax2

```
sp_iqaddlogin ‘username_in’, ‘pwd’, [ ’password_expiry_on_next_login ’]
[ , ’policy_name ’]
```

Syntax3

```
sp_iqaddlogin username_in, pwd, [ password_expiry_on_next_login ] [ ,
policy_name ]
```

Usage

Parameter	Description
username_in	The user’s login name. Login names must conform to the rules for identifiers

Parameter	Description
pwd	The user's password. Passwords must conform to rules for passwords, that is, they must be valid identifiers.
password_expiry_on_next_login	(Optional) Specifies whether user's password expires as soon as this user's login is created. Default setting is OFF (password does not expire).
policy_name	(Optional) Creates the user under the named login policy. If unspecified, user is created under the root login policy.

A username_in/pwd created using **sp_iqaddlogin** and set to expire in one day is valid all day tomorrow and invalid on the following day. In other words, a login created today and set to expire in *n* days are not usable once the date changes to the *(n+1)*th day.

Privileges

Requires the MANAGE ANY USER system privilege.

Description

Adds a new SAP Sybase IQ user account, assigns a login policy to the user and adds the user to the ISYSUSER system table. If the user already has a user ID for the database but is not in `ISYSUSER`, (for example, if the user ID was added using the **GRANT CONNECT** statement or Sybase Control Center), **sp_iqaddlogin** adds the user to the table.

If you do not specify a login policy name when calling the procedure, SAP Sybase IQ assigns the user to the root login policy.

Note: If the maximum number of logins for a login policy is unlimited, then a user belonging to that login policy can have an unlimited number of connections.

The first user login forces a password change and assigns a login policy to the newly created user. Use **CREATE USER** to create new users, although, for backward compatibility, **sp_iqaddlogin** is still supported.

Examples

These calls add the user `rose` with a password `irk324` under the login policy named `expired_password`. This example assumes the expired_password login policy already exists.

```
call sp_iqaddlogin('rose', 'irk324', 'ON', 'expired_password')
```

```
sp_iqaddlogin 'rose','irk324', 'ON', 'expired_password'
```

See also

- *sp_expireallpasswords system procedure* on page 668

- *sp_iqcopyloginpolicy Procedure* on page 459
- *sp_iqmodifylogin Procedure* on page 519
- *sp_iqpassword Procedure* on page 534
- *sp_iqdroplogin Procedure* on page 479

sp_iqbackupdetails Procedure

Shows all the dbfiles included in a particular backup.

Syntax

```
sp_iqbackupdetails   backup_id
```

Parameters

Table 70. Parameters

Parameter	Description
backup_id	Specifies the backup operation transaction identifier.

Note: You can obtain the backup_id value from the `SYSIQBACKUPHISTORY` table by executing the query:

```
select * from sysiqbackuphistory
```

Privileges

No specific system privileges are required to run this procedure.

Description

sp_iqbackupdetails returns:

Table 71. sp_iqbackupdetails Columns

Column Name	Description
backup_id	Identifier for the backup transaction.
backup_time	Time of the backup.
backup_type	Type of backup: “Full,” “Incremental since incremental,” or “Incremental since full.”
selective_type	Subtype of backup: "All inclusive," "All RW files in RW dbspaces," "Set of RO dbspace/file."
depends_on_id	Identifier for previous backup that the backup depends on.

Column Name	Description
dbspace_id	Identifier for the dbspace being backed up.
dbspace_name	Name of the dbspace from SYSIQBACKUPHISTORYDETAIL. If dbspace name matches the dbspace name in SYSDBSPACE for a given dbspace_id. Otherwise "null."
dbspace_rwstatus	"ReadWrite" or "Read Only."
dbspace_createid	Dbspace creation transaction identifier.
dbspace_alterid	Alter DBSPACE read-write mode transaction identifier.
dbspace_online	Status "Online" or "Offline."
dbspace_size	Size of dbspace, in KB, at time of backup.
dbspace_backup_size	Size of data, in KB, backed up in the dbspace.
dbfile_id	Identifier for the dbfile being backed up.
dbfile_name	The logical file name, if it was not renamed after the backup operation. If renamed, "null."
dbfile_rwstatus	"ReadWrite" or "Read Only."
dbfile_createid	Dbfile creation transaction identifier.
dbfile_alterid	Alter DBSPACE alter FILE read-write mode transaction identifier
dbfile_size in MB	Size of the dbfile, in KB.
dbfile_backup_size	Size of the dbfile backup, in KB.
dbfile_path	The dbfile path from SYSBACKUPDETAIL, if it matches the physical file path ("file_name") in SYSDBFILE for a given dbspace_id and the dbfile_id. Otherwise "null."

Example

Sample output from **sp_iqbackupdetails**:

```
backup_id   backup_time             backup_type   selective_type   d
epends_on_id
      883   2008-09-23 13:58:49.0   Full          All
inclusive                0

dbspace_id   dbspace_name   dbspace_rwstatus   dbspace_createid
         0   system         ReadWrite                         0

dbspace_alterid   dbspace_online dbspace_size dbspace_backup_size
dbfile_id
            0               0        2884                2884        0
```

```
dbfile_name dbfile_rwstatus dbfile_createid dbfile_alterid 
dbfile_size
    system       ReadWrite               0              0      2884
```

```
dbfile_backup_size dbfile_path
            2884  C:\\Documents and Settings\\All Users\\SybaseIQ\
\demo\\iqdemo.db
```

See also

- *SYSIQBACKUPHISTORY System View* on page 726

sp_iqbackupsummary Procedure

Summarizes backup operations performed.

Syntax

sp_iqbackupsummary [*timestamp or backup_id*]

Parameters

- **timestamp or backup_id –** specifies the interval for which to report backup operations. If you specify a timestamp or a backup ID, only those records with backup_time greater than or equal to the time you enter are returned. If you specify no timestamp, the procedure returns all the backup records in `ISYSIQBACKUPHISTORY`.

Privileges

No specific system privileges are required to run this procedure.

Description

Table 72. sp_iqbackupsummary Columns

Column Name	Description
backup_id	Identifier for the backup transaction
backup_time	Time of the backup
backup_type	Type of backup: “Full,” “Incremental since incremental,” or “Incremental since full”
selective_type	Subtype of backup: "All Inclusive," "All RW files in RW dbspaces," "Set of RO dbspace/file"
virtual_type	Type of virtual backup: “Non-virtual,” “Decoupled,” or “Encapsulated”

Column Name	Description
depends_on_id	Identifier for backup that the backup depends on
creator	Creator of the backup
backup_size	Size, in KB, of the backup
user_comment	User comment
backup_command	The backup statement issued (minus the comment)

Example

Sample output of **sp_iqbackupsummary**:

```
backup_id   backup_time             backup_type   selective_type   v
irtual_type
     883   2008-09-23 13:58:49.0   Full         All inclusive    Non
virtual

depends_on_id   creator   backup_size   user_comment   backup_command
           0   DBA               10864               backup database to
                                                              'c:\\\\temp
\\\\b1'
```

sp_iqcardinality_analysis Procedure

Analyzes the cardinality of columns in a table.

Note: sp_iqcardinality_analysis no longer returns an index type value or index recommendation. Users are advised to `Run Index Advisor` for suggestions about additional column indexes. **sp_iqcardinality_analysis** is deprecated and will be removed in a future release.

Syntax

sp_iqcardinality_analysis (['table_name'], ['table_owner'], ['script'])

Parameters

Parameter	Description
table_name	Name of the table.
table_owner	Name of the table owner. If this parameter is not specified, then the procedure looks for a table owned by the current user.

Parameter	Description
script	The script : • table_name • table_owner • column_name • cardinality • index_type • index recommendation

Usage

If you do not specify any parameters, then SAP Sybase IQ displays **create_index** SQL statements for all columns in all tables owned by the current user.

If you specify *script*, you can redirect the output to generate the script file:

```
OUTPUT TO ‘indexfile.sql’ FORMAT ASCII QUOTE '';
```

Privileges

Requires be either table owner or have SELECT ANY TABLE system privilege. In addition, user must have at least ONE of the following: CREATE ANY INDEX, ALTER ANY INDEX, CREATE ANY OBJECT, or ALTER ANY OBJECT.

Example

```
sp_iqcardinality_analysis 'Departments', 'GROUPO'
```

table_name	table_owner	column_name	cardinality	index type	Index Recommendation
Departments	GROUPO	DepartmentID	5		Run Index Advisor
Departments	GROUPO	DepartmentName	5		Run Index Advisor
Departments	GROUPO	DepartmentHeadID	5		Run Index Advisor

sp_iqcheckdb Procedure

Checks validity of the current database. Optionally corrects allocation problems for dbspaces or databases. **sp_iqcheckdb** does not check a partitioned table if partitioned data exists on offline dbspaces.

sp_iqcheckdb reads all storage in the database. On successful completion, the database free list (an internal allocation map) is updated to reflect the true storage allocation for the database. **sp_iqcheckdb** then generates a report listing the actions it has performed.

If an error is found, **sp_iqcheckdb** reports the name of the object and the type of error. **sp_iqcheckdb** does not update the free list if errors are detected.

sp_iqcheckdb also allows you to check the consistency of a specified table, index, index type, or the entire database.

Note: **sp_iqcheckdb** is the user interface to the SAP Sybase IQ database consistency checker (DBCC) and is sometimes referred to as **DBCC**.

Syntax

sp_iqcheckdb '*mode target* […] [resources *resource-percent*]'

This is the general syntax of **sp_iqcheckdb**. There are three modes for checking database consistency, and one for resetting allocation maps. The syntax for each mode is listed separately below. If mode and target are not both specified in the parameter string, SAP Sybase IQ returns the error message:

```
At least one mode and target must be specified to DBCC.
```

Parameters

mode: { **allocation** | **check** | **verify** } | **dropleaks**

target: [**indextype** *index-type* […]] **database** | **database resetclocks** | { [**indextype** *index-type*] […] **table** *table-name* [**partition** *partition-name*] […] | **index** *index-name* | […] **dbspace** *dbspace-name*}

Applies to

Simplex and multiplex.

Allocation Mode

sp_iqcheckdb 'allocation *target* [resources *resource-percent*]'

Check Mode

sp_iqcheckdb 'check *target* [resources *resource-percent*]'

Verify Mode

sp_iqcheckdb 'verify *target* [resources *resource-percent*]'

Dropleaks Mode

```
sp_iqcheckdb 'dropleaks target [ resources resource-percent ]'
```

Usage

Parameter	Description
database	If the target is a database, all dbspaces must be online.
index-type	One of the following index types: **FP**, **CMP**, **LF**, **HG**, **HNG**, **WD**, **DATE**, **TIME**, **DTTM**, **TEXT**. If the specified *index-type* does not exist in the target, an error message is returned. If multiple index types are specified and the target contains only some of these index types, the existing index types are processed by **sp_iqcheckdb**.
index-name	May contain owner and table qualifiers: `[[owner.]table-name.]index-name` If *owner* is not specified, current user and database owner (`dbo`) are substituted in that order. If *table* is not specified, *index-name* must be unique.
table-name	May contain an owner qualifier: `[owner.]table-name` If *owner* is not specified, current user and database owner (`dbo`) are substituted in that order. *table-name* cannot be a temporary or pre-join table. **Note:** If either the table name or the index name contains spaces, enclose the *table-name* or *index-name* parameter in double quotation marks: `sp_iqcheckdb 'check index "dbo.sstab.i2" resources 75'`

Parameter	Description
partition-name	The *partition-name* parameter contains no qualifiers. If it contains spaces, enclose it in double quotation marks. The partition filter causes **sp_iqcheckdb** to examine a subset of the corresponding table's rows that belong to that partition. A partition filter on a table and table target without the partition filter are semantically equivalent when the table has only one partition.
dbspace-name	The *dbspace-name* parameter contains no qualifiers. If it contains spaces, enclose it in double quotation marks. The dbspace target examines a subset of the database's pages that belong to that dbspace. The dbspace must be online. The dbspace and database target are semantically equivalent when the table has only one dbspace.
resource-percent	The input parameter *resource-percent* must be an integer greater than zero. The resources percentage allows you to limit the CPU utilization of the database consistency checker by controlling the number of threads with respect to the number of CPUs. If *resource-percent* = 100 (the default value), then one thread is created per CPU. If *resource-percent* > 100, then there are more threads than CPUs, which might increase performance for some machine configurations. The minimum number of threads is one.

Note: The **sp_iqcheckdb** parameter string must be enclosed in single quotes and cannot be greater than 255 bytes in length.

Allocation problems can be repaired in dropleaks mode.

Privileges

Requires the ALTER DATABASE system privilege. Users without the ALTER DATABASE system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

sp_iqcheckdb checks the allocation of every block in the database and saves the information in the current session until the next **sp_iqdbstatistics** procedure is issued. **sp_iqdbstatistics** displays the latest result from the most recent execution of **sp_iqcheckdb**.

sp_iqcheckdb can perform several different functions, depending on the parameters specified.

Allocation Mode

Checks allocation with blockmap information for the entire database, a specific index, a specific index type, a specific partition, specific table, or a specific dbspace. Does not check index consistency.

Detects duplicate blocks (blocks for which two or more objects claim ownership) or extra blocks (unallocated blocks owned by an object).

Detects leaked blocks (allocated blocks unclaimed by any object in the specified target) for database or dbspace targets.

When the target is a partitioned table, **allocation mode**:

- Checks metadata of all the table’s partition allocation bitmaps
- Checks metadata of the tables allocation bitmap
- Verifies that blockmap entries are consistent with the table’s allocation bitmap
- Verifies that none of the table’s partition allocation bitmaps overlap
- Checks that rows defined in the table’s partition allocation bitmaps form a superset of the table’s existence bitmap
- Checks that rows defined in the table’s partition allocation bitmaps form a superset of the table’s allocation bitmap

Note: sp_iqcheckdb cannot check all allocation problems if you specify the name of a single index, index type, or table in the input parameter string.

Run in allocation mode:

- To detect duplicate or unowned blocks (use database or specific tables or indexes as the target)
- If you encounter page header errors

The DBCC option **resetclocks** is used only with allocation mode. **resetclocks** is used with forced recovery to convert a multiplex secondary server to a coordinator. For information on multiplex capability, see *Using SAP Sybase IQ Multiplex*. **resetclocks** corrects the values of internal database versioning clocks, in the event that these clocks are behind. Do not use the **resetclocks** option for any other purpose, unless you contact SAP Sybase IQ Technical Support.

The **resetclocks** option must be run in single-user mode and is allowed only with the DBCC statement **allocation database**. The syntax of **resetclocks** is:

```
sp_iqcheckdb 'allocation database resetclocks'
```

Check Mode

Verifies that all database pages can be read for the entire database, specific index, specific index type, specific table, specific partition, or specific dbspace. If the table is partitioned, then check mode will check the table's partition allocation bitmaps.

Run in check mode if metadata, null count, or distinct count errors are returned when running a query.

Verify Mode

Verifies the contents of non-FP indexes with their corresponding FP indexes for the entire database, a specific index, a specific index type, specific table, specific partition, or specific dbspace. If the specified target contains all data pages for the FP and corresponding non-FP indexes, then verify mode detects the following inconsistencies:

- Missing key – a key that exists in the FP but not in the non-FP index.
- Extra key – a key that exists in the non-FP index but not in the FP index.
- Missing row – a row that exists in the FP but not in the non-FP index.
- Extra row – a row that exists in the non-FP index but not in the FP index.

If the specified target contains only a subset of the FP pages, then verify mode can detect only the following inconsistencies:

- Missing key
- Missing row

If the target is a partitioned table, then verify mode also verifies that each row in the table or table partition has been assigned to the correct partition.

Run in verify mode if metadata, null count, or distinct count errors are returned when running a query.

Note: **sp_iqcheckdb** does not check referential integrity or repair referential integrity violations.

Dropleaks Mode

When the SAP Sybase IQ server runs in single-node mode, you can use dropleaks mode with either a database or dbspace target to reset the allocation map for the entire database or specified dbspace targets. If the target is a dbspace, then the dropleaks operation must also prevent read-write operations on the named dbspace. All dbspaces in the database or dbspace list must be online.

On a multiplex coordinator node, dropleaks mode also detects leaked blocks, duplicate blocks, or extra blocks across the multiplex.

The following examples illustrate the use of the **sp_iqcheckdb** procedure.

Example 1
Check the allocation for the entire database:

```
sp_iqcheckdb 'allocation database'
```

Example 2
Perform a detailed check on indexes `i1`, `i2`, and `dbo.t1.i3`. If you do not specify a new mode, **sp_iqcheckdb** applies the same mode to the remaining targets, as shown in the following command:

```
sp_iqcheckdb 'verify index i1 index i2 index dbo.t1.i3'
```

Example 3
You can combine all modes and run multiple checks on a database in a single session. Perform a quick check of partition `p1` in table `t2`, a detailed check of index `i1`, and allocation checking for the entire database using half of the CPUs:

```
sp_iqcheckdb 'check table t2 partition p1 verify index i1
allocation database resources 50'
```

Example 4
Check all indexes of the type **FP** in the database:

```
sp_iqcheckdb 'check indextype FP database'
```

Example 5
Verify the **FP** and **HG** indexes in the table `t1` and the **LF** indexes in the table `t2`:

```
sp_iqcheckdb 'verify indextype FP indextype HG table t1 indextype LF
table t2'
```

Example 6
Check for LVC cell inconsistencies:

```
sp_iqcheckdb 'check index EFG2JKL.ASIQ_IDX_T208_C504_FP'
------------------------------------
Index Statistics:
** Inconsistent Index: abcd.EFG2JKL.ASIQ_IDX_T208_C504_FP ****** FP
Indexes Checked: 1
** Unowned LVC Cells: 212 ******
```

The **sp_iqcheckdb** LVC cells messages include:

- Unowned LVC cells
- Duplicate LVC cell rows
- Unallocated LVC cell rows

These messages indicate inconsistencies with a `VARCHAR`, `VARBINARY`, `LONG BINARY` (`BLOB`), or `LONG VARCHAR` (`CLOB`) column. Unowned LVC cells represent a small amount of unusable disk space and can safely be ignored. Duplicate and Unallocated LVC cells are serious errors that can be resolved only by dropping the damaged columns.

To drop a damaged column, create a new column from a copy of the old column, then drop the original column and rename the new column to the old column.

Note: LVC is a `VARCHAR` or `VARBINARY` column with a width greater than 255. `LONG BINARY` (`BLOB`) and `LONG VARCHAR` (`CLOB`) also use LVC.

DBCC performance

The execution time of DBCC varies, depending on the size of the database for an entire database check, the number of tables or indexes specified, and the size of the machine. Checking only a subset of the database (that is, only specified tables, indexes, or index types) requires less time than checking an entire database.

The processing time of **sp_iqcheckdb** dropleaks mode depends on the number of dbspace targets.

This table summarizes the actions and output of the four **sp_iqcheckdb** modes.

Table 73. Actions and Output of sp_iqcheckdb Modes

Mode	Errors Detected	Output	Speed
Allocation	Allocation errors	Allocation statistics only	4TB per hour
Check	Allocation errors Most index errors	All available statistics	60GB per hour
Verify	Allocation errors All index errors	All available statistics	15GB per hour
Dropleaks	Allocation errors	Allocation statistics only	4TB per hour

Output

Depending on the execution mode, **sp_iqcheckdb** output includes summary results, errors, informational statistics, and repair statistics. The output may contain as many as three results sets, if you specify multiple modes in a single session. Error statistics are indicated by asterisks (*****), and appear only if errors are detected.

The output of **sp_iqcheckdb** is also copied to the SAP Sybase IQ message file `.iqmsg`. If the **DBCC_LOG_PROGRESS** option is ON, **sp_iqcheckdb** sends progress messages to the IQ message file, allowing the user to follow the progress of the DBCC operation as it executes.

Output Example

Run **sp_iqcheckdb 'allocation database'** :

```
=====================================================================
DBCC Allocation Mode Report
=====================================================================
   DBCC Status                                      No Errors Detected
=====================================================================
Allocation Summary
===================================================================

   Blocks Total                                     25600
   Blocks in Current Version                        5917
   Blocks in All Versions                           5917
   Blocks in Use                                    5917
   % Blocks in Use                                  23
===================================================================
Allocation Statistics
===================================================================
   Marked Logical Blocks                            8320
   Marked Physical Blocks                           5917
   Marked Pages                                     520
   Blocks in Freelist                               2071196
   Imaginary Blocks                                 2014079
   Highest PBN in Use                               1049285
   Total Free Blocks                                19683
   Usable Free Blocks                               19382
   % Total Space Fragmented                         1
   % Free Space Fragmented                          1
   Max Blocks Per Page                              16
   1  Block Page Count                              165
   3  Block Page Count                              200
   4  Block Page Count                              1
   10 Block Page Count                              1
   16 Block Page Count                              153
   2  Block Hole Count                              1
   3  Block Hole Count                              19
   6  Block Hole Count                              12
   7  Block Hole Count                              1
   10 Block Hole Count                              1
   15 Block Hole Count                              1
   16 Block Hole Count                              1220

Partition Summary
   Database Objects Checked                         2
   Blockmap Identity Count                          2
   Bitmap Count                                     2
===================================================================
Connection Statistics
===================================================================
   Sort Records                                     3260
   Sort Sets                                        2
===================================================================


DBCC Info
===================================================================
   DBCC Work units Dispatched                       197
   DBCC Work units Completed                        197
   DBCC Buffer Quota                                255
   DBCC Per-Thread Buffer Quota                     255
   Max Blockmap ID found                            200
   Max Transaction ID found                         404
```

Note: The report may indicate leaked space. Leaked space is a block that is allocated according to the database free list (an internal allocation map), but DBCC finds that the block is not part of any database object.

sp_iqcheckoptions Procedure

For the connected user, **sp_iqcheckoptions** displays a list of the current value and the default value of database and server startup options that have been changed from the default.

Syntax

```
sp_iqcheckoptions
```

Privileges

None. The DBA sees all options set on a permanent basis for all roles and users and sees temporary options set for DBA. Users who are not DBAs see their own temporary options. All users see nondefault server startup options.

Usage

Requires no parameters. Returns one row for each option that has been changed from the default value. The output is sorted by option name, then by user name.

Description

For the connected user, the **sp_iqcheckoptions** stored procedure displays a list of the current value and the default value of database and server startup options that have been changed from the default. **sp_iqcheckoptions** considers all SAP Sybase IQ and SQL Anywhere database options. SAP Sybase IQ modifies some SQL Anywhere option defaults, and these modified values become the new default values. Unless the new SAP Sybase IQ default value is changed again, **sp_iqcheckoptions** does not list the option.

When **sp_iqcheckoptions** is run, the DBA sees all options set on a permanent basis for all roles and users and sees temporary options set for DBA. Users who are not DBAs see their own temporary options. All users see nondefault server startup options.

Table 74. sp_iqcheckoptions Columns

Column Name	Description
User_name	The name of the user or role for whom the option has been set. At database creation, all options are set for the `PUBLIC` role. Any option that has been set for a role or user other than `PUBLIC` is displayed.
Option_name	The name of the option.
Current_value	The current value of the option.
Default_value	The default value of the option.

Column Name	Description
Option_type	“Temporary” for a TEMPORARY option, else “Permanent”.

Examples

In these examples, the temporary option `APPEND_LOAD` is set to ON and the role `myrole` has the option `MAX_WARNINGS` set to 9. The user `joel` has a temporary value of 55 set for `MAX_WARNINGS`.

In the first example, **sp_iqcheckoptions** is run by the DBA.

```
User_name Option_name          Current_value   Default_value   Optio
n_type
DBA      Ansi_update_constr   CURSORS        Off             Permanent
PUBLIC    Ansi_update_constr   Cursors        Off             Permanent
DBA      Checkpoint_time      20             60              Temporary
DBA      Connection_authent   Company=MyComp;                Temporary
                               Application=DBTools;Signa
DBA       Login_procedure      DBA.sp_iq_proce sp_login_envir  Perma
nent
PUBLIC    Login_procedure      DBA.sp_iq_proce sp_login_envir  Perma
nent
myrole    Max_Warnings        9             281474976710655 Permanent
DBA       Thread_count        25              0              Temporary
```

In the second example, **sp_iqcheckoptions** is run by the user `joel`.

```
User_name Option_name          Current_value   Default_value   Optio
n_type
joel      Ansi_update_constr   CURSORS        Off             Permanent
PUBLIC    Ansi_update_constr   Cursors        Off             Permanent
joel      Checkpoint_time      20             60              Temporary
joel      Connection_authent   Company=MyComp;                Temporary
                               Application=DBTools;Signa
joel      Login_procedure      DBA.sp_iq_proce sp_login_envir  Perma
nent
PUBLIC    Login_procedure      DBA.sp_iq_proce sp_login_envir  Perma
nent
joel      Max_Warnings        55            281474976710655 Temporary
joel      Thread_count        25              0              Temporary
```

sp_iqclient_lookup Procedure

Allows a client application to determine the SAP Sybase IQ user account responsible for a particular data stream, as observed in a network analyzer originating from a specific client IP address/port.

Syntax

sp_iqclient_lookup `[ 'IPaddress' ], [ Port ], [ UserID ]`

Parameters

Parameter	Description
IPaddress	Specifies the IP address of the originating client application.
Port	Specifies the port number of the originating client application.
UserID	Specifies the SAP Sybase IQ user ID.

Privileges

Requires the MONITOR, DROP CONNECTION, or SERVER OPERATOR system privilege. Users without one of these system privileges must be granted `EXECUTE` permission to run the stored procedure.

Description

The **sp_iqclient_lookup** procedure takes the client IP address and port number and returns a single row containing Number (the connection ID), IPaddress, Port, and UserID.

```
1> sp_iqclient_lookup '158.76.235.71',3360
2> go

Number   IPaddress      Port    UserID
------   ---------      ----    ------
15       158.76.235.71  3360    rdeniro
```

Optionally, you can pass a third argument to select only the UserID. If no arguments are passed, sp_iqclient_lookup returns all current logins with their IP addresses and port numbers. For example:

```
sp_iqclient_lookup
```

```
Number   IPaddress        Port    UserID
------   ---------        ----    ------
11       162.66.131.36    2082    mbrando
21       162.66.100.233   1863    apacino
22       162.66.100.206   8080    jcaan
23       162.66.100.119   6901    rduvall
24       162.66.100.125   7001    dkeaton
25       162.66.100.124   6347    jcazale

(6 rows affected)
(return status = 0)
```

If a client application is not using TCP/IP or for internal connections, the address appears as 127.0.0.1.

Note: This information is available for logged on users only. No historical login data is kept on the server for this purpose.

Side Effects

The **sp_iqclient_lookup** stored procedure may impact server performance, which varies from one installation to another. Finding the login name entails scanning through all current active connections on the server; therefore, the impact may be greater on servers with large numbers of connections. Furthermore, this information cannot be cached as it is dynamic — sometimes highly dynamic. It is, therefore, a matter for the local system administrator to manage the use of this stored procedure, as well as monitor the effects on the server, just as for any other client application that uses server facilities.

Examples

Shows IP addresses for UserID jcazale:

```
sp_iqclient_lookup null, null, jcazale
```

```
Number   IPaddress        Port    UserID
------   ----------       ----    ------
```

```
11       162.66.131.36    2082    jcazale
15       164.66.131.36    1078    jcazale
```

Shows IP addresses from client IP 162.66.131.36:

```
sp_iqclient_lookup '162.66.131.36'
```

```
Number   IPaddress        Port    UserID
------   ----------       ----    ------
11       162.66.131.36    2082    jcazale
12       162.66.131.36    1078    jcaan
```

Note: The result is empty when the user specifies an incorrect argument.

sp_iqcolumn Procedure

Displays information about columns in a database.

Syntax1

sp_iqcolumn ([*table_name*],[*table_owner*], [*table_loc*])

Syntax2

sp_iqcolumn [table_name='*table_name*'],[table_owner='*tableowner*'], [table_loc='*table_loc*']

Usage

Syntax	Description
Syntax1	If you specify *table_owner* without specifying *table_name*, you must substitute NULL for *table_name*. For example, `sp_iqcolumn NULL,DBA`.
Syntax2	The parameters can be specified in any order. Enclose *'table_name'* and *'table_owner'* in single quotes.

Privileges

No specific system privileges are required to run this stored procedure.

Description

Displays information about columns in a database. Specifying the *table_name* parameter returns the columns only from tables with that name. Specifying the table_owner parameter returns only tables owned by that user. Specifying both *table_name* and table_owner parameters chooses the columns from a unique table, if that table exists. Specifying table_loc returns only tables that are defined in that segment type. Specifying no parameters returns all columns for all tables in a database. **sp_iqcolumn** does not return column information for system tables.

Column name	Description
table_name	The name of the table
table_owner	The owner of the table
column_name	The name of the column
domain_name	The data type
width	The precision of numeric data types that have precision and scale or the storage width of numeric data types without scale; the width of character data types
scale	The scale of numeric data types
nulls	'Y' if the column can contain NULLS, 'N' if the column cannot contain NULLS
default	'Identity/Autoincrement' if the column is an identity/autoincrement column, null if not.

Column name	Description
cardinality	The distinct count, if known, by indexes.
location	TEMP = IQ temporary store, MAIN = IQ main store, SYSTEM = catalog store
isPartitioned	'Y' if the column belongs to a partitioned table and has one or more partitions whose dbspace is different from the table partition’s dbspace, 'N' if the column’s table is not partitioned or each partition of the column resides in the same dbspace as the table partition.
remarks	User comments added with the **COMMENT** statement
check	the check constraint expression

See also

- *sp_iqconstraint Procedure* on page 455
- *sp_iqdatatype Procedure* on page 463
- *sp_iqevent Procedure* on page 484
- *sp_iqhelp Procedure* on page 489
- *sp_iqindex and sp_iqindex_alt Procedures* on page 496
- *sp_iqpkeys Procedure* on page 536
- *sp_iqprocparm Procedure* on page 540
- *sp_iq_reset_identity Procedure* on page 549
- *sp_iqtable Procedure* on page 575
- *sp_iqview Procedure* on page 590

sp_iqcolumn Procedure Example

Use the example as a reference for **sp_iqcolumn** usage.

The following variations in syntax both return all of the columns in the table `Departments`:

```
sp_iqcolumn Departments
```

```
call sp_iqcolumn (table_name='Departments')
```

```
table_name   table_owner  column_name     domain_name  width  scale
  nulls  default
Departments  GROUPO       DepartmentID    integer       4        0
  N       (NULL)
Departments  GROUPO       DepartmentName  char         40        0
  N       (NULL)
Departments  GROUPO       DepartmentHead  integer       4        0
  Y       (NULL)

cardinality  location  isPartitioned  remarks  check
5            Main       N              (NULL)   (NULL)
```

```
0            Main      N            (NULL)   (NULL)
0            Main      N            (NULL)   (NULL)
```

The following variation in syntax returns all of the columns in all of the tables owned by table owner `DBA`.

```
sp_iqcolumn table_owner='DBA'
```

sp_iqcolumnmetadata Procedure

Returns details about column indexes in one or more tables.

Syntax

sp_iqcolumnmetadata *[table.name [, owner-name]]*

Privileges

User must be a table owner, have REFERENCE permissions on the table, or have either the ALTER ANY INDEX or ALTER ANY OBJECT system privilege..

Description

sp_iqcolumnmetadata reads the index metadata to return details about column indexes in both base and global temporary tables. Index metadata reported for a global temporary table is for the individual instance of that table.

Include the optional *[table.name]* parameter to generate details for that table. Omit the *[table.name]* parameter to generate details for all tables in the database.

sp_iqcolumnuse Procedure

Reports detailed usage information for columns accessed by the workload.

Syntax

```
sp_iqcolumnuse
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

Columns from tables created in SYSTEM are not reported.

Table 75. sp_iqcolumnuse columns

Column name	Description
TableName	Table name
ColumnName	Column name

Column name	Description
Owner	User name of column owner
UID**	Column Unique Identifier
LastDT	Date/time of last access
NRef	Number of query references

**UID is a number assigned by the system that uniquely identifies the instance of the column (where instance is defined when an object is created).

Tip: The INDEX_ADVISOR option generates messages suggesting additional column indexes that may improve performance of one or more queries.

Example

Sample output from the **sp_iqcolumnuse** procedure:

```
TableName       ColumnName     Owner   UID             LastDT
         NRef
orders          o_orderdate    DBA                151   20070917
22:41:22..  13
orders          o_shippriority DBA                154   20070917
22:41:22..  13
lineitem        l_orderkey     DBA                186   20070917
22:41:22..  13
lineitem        l_extendedp..  DBA                191   20070917
22:41:22..  13
lineitem        l_discount     DBA                192   20070917
22:41:22..  13
lineitem        l_shipdate     DBA                196   20070917
22:41:22..  13
#tmp1           expression     DBA     10000000001218   20070917
22:57:36..   1
#tmp1           expression     DBA     10000000001222   20070917
22:41:58..   1
...
```

Note: The long numbers in the example above are temporary IDs.

See also

- *sp_iqindexadvice Procedure* on page 499
- *sp_iqindexuse Procedure* on page 512
- *sp_iqtableuse Procedure* on page 580
- *sp_iqunusedcolumn Procedure* on page 585
- *sp_iqunusedindex Procedure* on page 586
- *sp_iqunusedtable Procedure* on page 587
- *sp_iqworkmon Procedure* on page 596

sp_iqconnection Procedure

Shows information about connections and versions, including which users are using temporary dbspace, which users are keeping versions alive, what the connections are doing inside SAP Sybase IQ, connection status, database version status, and so on.

Syntax

```
sp_iqconnection [ connhandle ]
```

Applies to

Simplex and multiplex.

Usage

connhandle is equal to the `Number` connection property and is the ID number of the connection. The **connection_property** system function returns the connection ID:

```
SELECT connection_property ( 'Number' )
```

When called with an input parameter of a valid *connhandle*, **sp_iqconnection** returns the one row for that connection only.

Privileges

Requires the DROP CONNECTION, MONITOR or SERVER OPERATOR system privilege. Users without one of these system privileges must be granted `EXECUTE` permission to run the stored procedure.

Description

sp_iqconnection returns a row for each active connection. The columns ConnHandle, Name, Userid, LastReqTime, ReqType, CommLink, NodeAddr, and LastIdle are the connection properties Number, Name, Userid, LastReqTime, ReqType, CommLink, NodeAddr, and LastIdle respectively, and return the same values as the system function **sa_conn_info**. The additional columns return connection data from the SAP Sybase IQ side of the SAP Sybase IQ engine. Rows are ordered by ConnCreateTime.

The column MPXServerName stores information related to internode communication (INC), as shown:

Server Where Run	MPXServerName Column Content
Simplex server	NULL (All connections are local/user connections)

Server Where Run	MPXServerName Column Content
Multiplex coordinator	• NULL for local/user connections • Contains value of secondary node's server name (source of connection) for every INC connection (either on-demand or dedicated heartbeat connection)
Multiplex secondary	• NULL for local/user connections • Contains value of coordinator's server name (source of connection).

In Java applications, specify SAP Sybase IQ-specific connection properties from TDS clients in the RemotePWD field. This example, where **myconnection** becomes the IQ connection name, shows how to specify IQ specific connection parameters:

```
p.put("RemotePWD",",,CON=myconnection");
```

Column Name	Description
ConnHandle	The ID number of the connection.
Name	The name of the server.
Userid	The user ID for the connection.
LastReqTime	The time at which the last request for the specified connection started.
ReqType	A string for the type of the last request.
IQCmdType	The current command executing on the SAP Sybase IQ side, if any. The command type reflects commands defined at the implementation level of the engine. These commands consist of transaction commands, DDL and DML commands for data in the IQ store, internal IQ cursor commands, and special control commands such as **OPEN** and **CLOSE DB**, **BACKUP**, **RESTORE**, and others.
LastIQCmdTime	The time the last IQ command started or completed on the IQ side of the SAP Sybase IQ engine on this connection.
IQCursors	The number of cursors open in the IQ store on this connection.

Column Name	Description
LowestIQCursorState	The IQ cursor state, if any. If multiple cursors exist on the connection, the state that appears is the lowest cursor state of all the cursors; that is, the furthest from completion. Cursor state reflects internal SAP Sybase IQ implementation detail and is subject to change in the future. For this version, cursor states are: NONE, INITIALIZED, PARSED, DESCRIBED, COSTED, PREPARED, EXECUTED, FETCHING, END_OF_DATA, CLOSED and COMPLETED. As suggested by the names, cursor state changes at the end of the operation. A state of PREPARED, for example, indicates that the cursor is executing.
IQthreads	The number of SAP Sybase IQ threads currently assigned to the connection. Some threads may be assigned but idle. This column can help you determine which connections are using the most resources.
TxnID	The transaction ID of the current transaction on the connection. This is the same as the transaction ID in the `.iqmsg` file by the BeginTxn, CmtTxn, and PostCmtTxn messages, as well as the Txn ID Seq logged when the database is opened.
ConnCreateTime	The time the connection was created.
TempTableSpaceKB	The number of kilobytes of IQ temporary store space in use by this connection for data stored in IQ temp tables.
TempWorkSpaceKB	The number of kilobytes of IQ temporary store space in use by this connection for working space such as sorts, hashes, and temporary bitmaps. Space used by bitmaps or other objects that are part of indexes on SAP Sybase IQ temporary tables are reflected in TempTableSpaceKB.
IQConnID	The ten-digit connection ID included as part of all messages in the `.iqmsg` file. This is a monotonically increasing integer unique within a server session.
satoiq_count	An internal counter used to display the number of crossings from the SQL Anywhere side to the IQ side of the SAP Sybase IQ engine. This might be occasionally useful in determining connection activity. Result sets are returned in buffers of rows and do not increment satoiq_count or iqtosa_count once per row.
iqtosa_count	An internal counter used to display the number of crossings from the IQ side to the SQL Anywhere side of the SAP Sybase IQ engine. This might be occasionally useful in determining connection activity.
CommLink	The communication link for the connection. This is one of the network protocols supported by SAP Sybase IQ, or is local for a same-machine connection.
NodeAddr	The node for the client in a client/server connection.
LastIdle	The number of ticks between requests.

Column Name	Description
MPXServerName	If an INC connection, the varchar(128) value contains the name of the multiplex server where the INC connection originates. NULL if not an INC connection.
LSName	The logical server name of the connection. NULL if logical server context is unknown or not applicable.
INCConnName	The name of the underlying INC connection for a user connection. The data type for this column is varchar(255). If **sp_iqconnection** shows an INC connection name for a suspended user connection, that user connection has an associated INC connection that is also suspended.
INCConnSuspended	The value "Y" in this column indicates that the underlying INC connection for a user connection is in a suspended state. The value "N" indicates that the connection is not suspended.

Example

sp_iqconnection

```
ConnHandle      Name    Userid                 LastReqTime   ReqType
=== =================== ======  ==========================   =======
 1  'SQL_DBC_100525210' 'DBA'   '2011-03-28  09:29:24.466'  'OPEN'


           IQCmdType    LastIQCmdTime      IQCursors  LowestIQCursorState
====================    ==============     =========  ===================
'IQUTILITYOPENCURSOR'  2011-03-28 09:29:24.0       0             'NONE'


IQthreads   TxnID         ConnCreateTime TempTableSpaceKB TempWorkSpaceKB
=========  =======  =====================  =============== ===============
        0  3352568  2011-03-28 09:29:20.0               0           0


IQconnID satoiq_count iqtosa_count CommLink NodeAdd LastIdle MPXServerName
======== ============ ============ ======== ======= ======== ============
      34           43          2  'local'      ''      244  (NULL)

LSName      INCConnName                  INCConnSuspended
==========  ===================          ================
Finance_LS  'IQ_MPX_SERVER_P54'          'Y'
```

sp_iqconstraint Procedure

Lists referential integrity constraints defined using **CREATE TABLE** or **ALTER TABLE** for the specified table or column.

Syntax

```
sp_iqconstraint [ 'table-name', 'column-name', 'table-owner' ]
```

Privileges

No specific system privilege is required to run this procedure.

Description

If table name and column name are omitted, reports all referential integrity constraints for all tables including temporary ones in the current connected database. The information includes unique or primary key constraint, referential constraint, and associated role name that are defined by the **CREATE TABLE** and/or **ALTER TABLE** statements.

Example

This is sample output that displays all primary key/foreign key pairs where either the candidate key or foreign key contains column `ck1` for owner `bob` in all tables:

```
call sp_iqconstraint('','ck1','bob')
```

```
PTAB1 bob ASIQ_IDX_T27_HG  unique   ck1,ck2  selftab bob CK6FK3  Y
ASIQ_IDX_T42_HG  ck1,ck2PTAB2 bob ASIQ_IDX_T27_HG  unique   ck1,ck2
selftab bob CK6FK4  Y
ASIQ_IDX_T206_I42_HG  ck1,ck2selftab bob ASIQ_IDX_T26_HG  unique
ck1,ck2  selftab bob CK3FK1  Y
ASIQ_IDX_T206_I42_HG  ck1,ck2
```

The columns displayed are:

- Primary enforced table
- Table owner
- Candidate key index
- Primary key or Unique
- Primary key columns
- Foreign table
- Foreign table owner
- Foreign key role name
- Enforced status ("Y" for enforced, "N" for unenforced)
- Foreign key index
- Foreign key columns
- Location ("TEMP," "MAIN," or "SYSTEM")

See also

- *sp_iqcolumn Procedure* on page 447
- *sp_iqdatatype Procedure* on page 463
- *sp_iqevent Procedure* on page 484
- *sp_iqhelp Procedure* on page 489
- *sp_iqindex and sp_iqindex_alt Procedures* on page 496
- *sp_iqpkeys Procedure* on page 536
- *sp_iqprocparm Procedure* on page 540
- *sp_iq_reset_identity Procedure* on page 549
- *sp_iqtable Procedure* on page 575

- *sp_iqview Procedure* on page 590

sp_iqcontext Procedure

Tracks and displays, by connection, information about statements that are currently executing.

Syntax

```
sp_iqcontext [ connhandle ]
```

Usage

The input parameter *connhandle* is equal to the `Number` connection property and is the ID number of the connection. For example, **SELECT CONNECTION_PROPERTY('NUMBER')**.

When called with an input parameter of a valid *connhandle*, **sp_iqcontext** returns the information only for that connection.

Privileges

Requires the MANAGE ANY USER or MONITOR system privileges.

Description

sp_iqcontext lets the DBA determine what statements are running on the system at any given moment, and identify the user and connection that issued the statement. With this information, you can use this utility to:

- Match the statement text with the equivalent line in **sp_iqconnection** to get resource usage and transactional information about each connection
- Match the statement text to the equivalent line in the SQL log created when the **-zr** server option is set to ALL or SQL
- Use connection information to match the statement text in **sp_iqcontext** to the equivalent line in the `.iqmsg` file, which includes the query plan, when SAP Sybase IQ can collect it
- Match statement text to an SAP Sybase IQ stack trace (`stktrc-yyyymmdd-hhnnss_#.iq`), if one is produced
- Collate this information with an operating system stack trace that might be produced, such as pstack on Sun Solaris

The maximum size of statement text collected is the page size of the catalog store.

Table 76. sp_iqcontext columns

Column Name	Description
ConnOrCursor	CONNECTION, CURSOR, or DQP.
ConnHandle	The ID number of the connection or 0 for DQP.
Name	The name of the server (leader name).

Column Name	Description
Userid	The user ID for the connection, cursor, or DQP worker.
numIQCursors	If column 1 is CONNECTION, the number of cursors open on this connection. If column 1 is CURSOR, a number assigned sequentially to cursors associated with this connection. If column 1 is DQP, then 0. CONNECTION can also return a value of 0.
IQthreads	The number of IQ threads currently assigned to the connection. Some threads may be assigned but idle. For DQP threads, indicates the number of threads assigned to the DQP worker.
TxnID	The transaction ID of the current transaction. In the case of a worker thread, indicates the leader's transaction ID.
ConnOrCurCreateTime	The time this connection, cursor, or DQP worker was created.
IQConnID	The connection ID displayed as part of all messages in the `.iqmsg` file. This is a monotonically increasing integer unique within a server session.
IQGovernPriority	A value that indicates the order in which the queries of a user are queued for execution. 1 indicates high priority, 2 (the default) medium priority, and 3 low priority. A value of -1 indicates that IQGovernPriority does not apply to the operation. Set the IQGovernPriority value with the database option `IQGOVERN_PRIORITY`. For DQP connections, this column displays `No command`.
CmdLine	First 4096 characters of the user command being executed. For DQP connections, this column displays `No command`.
Attributes	Unique ID for the query being distributed.

Example

The following example shows an excerpt from output when `sp_iqcontext` is issued with no parameter, producing results for all current connections. Column names are truncated due to space considerations.

```
ConnOrCu.. ConnHandle Name UserId numIQ.. IQthr.. TxnID Conn.. IQcon..
IQGov.. Cmd.. Attributes
CONNECTION 2 sun7bar dbo 0 0 0 2010-08-04 15:15:40.0 15 No command NO
COMMAND
```

```
CONNECTION 7 sun7bar dbo 0 0 0 2010-08-04 15:16:00.0 32 No command NO
COMMAND
CONNECTION 10 sun7bar dbo 0 0 0 2010-08-04 15:16:21.0 46 No command NO
COMMAND
...
CONNECTION 229 sun7bar DBA 0 0 1250445 2010-08-05 18:28:16.0 50887 2
select server_name,
inc_state, coordinator_failover from sp_iqmpxinfo() order by server_name
...
DQP 0 dbsrv2873_node_c1DBA 0 1 10000 2010-08-05 18:28:16.0 no command no
command Query ID:
12345; Condition: c1 > 100;
DQP 0 dbsrv2873_node_c1DBA 0 1 10001 2010-08-05 18:28:16.0 no command no
command Query ID:
12346; Node #12 Join (Hash);
```

The first line of output shows connection 2 (IQ connection ID 15). This connection is on server sun7bar, user dbo. This connection was not executing a command when `sp_iqcontext` was issued.

Connection 229 shows the user command being executed (the command contains less than the maximum 4096 characters the column can display). The `2` before the user command fragment indicates that this is a medium priority query.

The connection handle (2 for the first connection in this example) identifies results in the **-zr** log. The IQ connection ID (15 for the first connection in this example) identifies results in the `.iqmsg` file. On UNIX systems, you can use **grep** to locate all instances of the connection handle or connection ID, making it easy to correlate information from all sources.

The second-last line (TxnID `10000`) shows a DQP worker thread. The worker connection is running two invariant conditions.

The last line (TxnID `10001`) shows connection is running a hash join.

See also

- *CONNECTION_PROPERTY Function [System]* on page 176
- *sp_iqshowpsexe Procedure* on page 557

sp_iqcopyloginpolicy Procedure

Creates a new login policy by copying an existing one.

Syntax1

call sp_iqcopyloginpolicy (*'existing-policy-name'*, *'new-policy-name'*)

Syntax2

sp_iqcopyloginpolicy *'existing-policy-name'*, *'new-policy-name'*

Usage

Table 77. Parameters

Parameter	Description
existing policy name	The login policy to copy.
new policy name	Name of the new login policy to create (CHAR(128)).

Privileges

Requires the MANAGE ANY LOGIN POLICY system privilege.

Examples

Creates a new login policy named *lockeduser* by copying the login policy option values from the existing login policy named *root*:

```
call sp_iqcopyloginpolicy ('root','lockeduser')
```

See also

- *sp_expireallpasswords system procedure* on page 668
- *sp_iqaddlogin Procedure* on page 429
- *sp_iqmodifylogin Procedure* on page 519
- *sp_iqpassword Procedure* on page 534

sp_iqcursorinfo Procedure

Displays detailed information about cursors currently open on the server.

Syntax

```
sp_iqcursorinfo [ cursor-name ] [, conn-handle ]
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Usage

Parameter	Description
cursor-name	The name of the cursor. If only this parameter is specified, **sp_iqcursorinfo** returns information about all cursors that have the specified name in all connections. The login policy to copy.
conn-handle	An integer representing the connection ID. If only this parameter is specified, **sp_iqcursorinfo** returns information about all cursors in the specified connection.

The **sp_iqcursorinfo** procedure can be invoked without any parameters. If no parameters are specified, **sp_iqcursorinfo** returns information about all cursors currently open on the server. If both parameters are specified, **sp_iqcursorinfo** reports information about all of the cursors that have the specified name and are in the specified connection.

If you do not specify the first parameter, but specify the second parameter, you must substitute NULL for the omitted parameter. For example, `sp_iqcursorinfo NULL, 1`.

Table 78. sp_iqcursorinfo usage examples

Syntax	Output
sp_iqcursorinfo	Displays information about all cursors currently open on the server
sp_iqcursorinfo ‘cursor1’	Displays information about the all cursors named `cursor1` in all connections
sp_iqcursorinfo NULL, 3	Displays information about all cursors in connection 3
sp_iqcursorinfo ‘cursor2’, 4	Displays information about all the cursors named `cursor2` in connection 4

Description

The **sp_iqcursorinfo** stored procedure displays detailed information about cursors currently open on the server. The **sp_iqcursorinfo** procedure enables database administrators to monitor cursor status using just one stored procedure and view statistics such as how many rows have been updated, deleted, and inserted.

If you specify one or more parameters, the result is filtered by the specified parameters. For example, if *cursor-name* is specified, only information about the specified cursor is displayed. If *conn-handle* is specified, **sp_iqcursorinfo** returns information only about cursors in the specified connection. If no parameters are specified, **sp_iqcursorinfo** displays information about all cursors currently open on the server.

The **sp_iqcursorinfo** procedure returns information in the following columns:

Table 79. sp_iqcursorinfo columns

Column name	Description
Name	The name of the cursor
ConnHandle	The ID number of the connection
IsUpd	Y: the cursor is updatable; N otherwise
IsHold	Y: the cursor is a hold cursor; N otherwise
IQConnID	The ten digit connection ID displayed as part of all messages in the `.iqmsg` file. This number is a monotonically increasing integer unique within a server session.
UserID	User ID (or user name) for the user who created and ran the cursor
CreateTime	The time of cursor creation
CurrentRow	The current position of the cursor in the result set
NumFetch	The number of times the cursor fetches a row. The same row can be fetched more than once.
NumUpdate	The number of times the cursor updates a row, if the cursor is updatable. The same row can be updated more than once.
NumDelete	The number of times the cursor deletes a row, if the cursor is updatable.
NumInsert	The number of times the cursor inserts a row, if the cursor is updatable.
RWTabOwner	The owner of the table that is opened in RW mode by the cursor.
RWTabName	The name of the table that is opened in RW mode by the cursor.
CmdLine	The first 4096 characters of the command the user executed

Example

Display information about all cursors currently open on the server:

```
sp_iqcursorinfo

Name      ConnHandle      IsUpd      IsHold      IQConnID      UserID
-------------------------------------------------------------------
--
crsr1              1          Y           N           118         DBA
crsr2              3          N           N           118         DBA
```

```
CreateTime                  CurrentRow      NumFetch      NumUpdate
-----------------------------------------------------------------
2009-06-26 15:24:36.000          19      100000000     200000000
2009-06-26 15:38:38.000       20000      200000000

NumDelete     NumInsert      RWTabOwner      RWTabName        CmdLine
------------------------------------------------------------------
---
 20000000    3000000000            DBA         test1    call proc1()
                                                        call proc2()
```

sp_iqdatatype Procedure

Displays information about system data types and user-defined data types.

Syntax

sp_iqdatatype [*type-name*], [*type-owner*], [*type-type*]

Privileges

No specific system privileges are required to run this procedure.

Usage

Table 80. Parameters

Parameter	Description
type-name	The name of the data type.
type-owner	The name of the creator of the data type.
type-type	The type of data type. Allowed values are: • **SYSTEM**: displays information about system defined data types (data types owned by user `SYS` or `dbo`) only • **ALL**: displays information about user and system data types • Any other value: displays information about user data types

The **sp_iqdatatype** procedure can be invoked without any parameters. If no parameters are specified, only information about user-defined data types (data types not owned by `dbo` or `SYS`) is displayed by default.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqdatatype NULL, NULL, SYSTEM` and `sp_iqdatatype NULL, user1`.

Table 81. sp_iqdatatype usage examples

Syntax	Output
sp_iqdatatype	Displays information about all user-defined data types in the database
sp_iqdatatype country_t	Displays information about the user-defined data type named `country_t`
sp_iqdatatype non_existing_type	No rows returned, as the data type `non_existing_type` does not exist
sp_iqdatatype NULL, DBA	Displays information about all user-defined data types owned by `DBA`
sp_iqdatatype country_t, DBA	Displays information about the data type `country_t` owned by `DBA`
sp_iqdatatype rowid	`rowid` is a system-defined data type. If there is no user-defined data type also named `rowid`, no rows are returned. (By default, only user-defined data types are returned.)
sp_iqdatatype rowid, SYS	No rows returned, as the data type `rowid` is not a user-defined data type (by default, only user-defined data types are returned)
sp_iqdatatype NULL, NULL, SYSTEM	Displays information about all system defined data types (owned by `dbo` or `SYS`)
sp_iqdatatype rowid, NULL, SYSTEM	Displays information about the system data type `rowid`
sp_iqdatatype NULL, NULL, 'ALL'	Displays information about the user-defined and system data types

Description

The **sp_iqdatatype** stored procedure displays information about system and user-defined data types in a database. User-defined data types are also referred to as domains. Predefined domain names are not included in the **sp_iqdatatype** output.

If you specify one or more parameters, the **sp_iqdatatype** result is filtered by the specified parameters. For example, if *type-name* is specified, only information about the specified data type is displayed. If *type-owner* is specified, **sp_iqdatatype** only returns information about data types owned by the specified owner. If no parameters are specified, **sp_iqdatatype** displays information about all the user-defined data types in the database.

The **sp_iqdatatype** procedure returns information in the following columns:

Table 82. sp_iqdatatype columns

Column name	Description
type_name	The name of the data type
creator	The owner of the data type
nulls	Y indicates the user-defined data type allows nulls; N indicates the data type does not allow nulls and U indicates the null value for the data type is unspecified.
width	Displays the length of string columns, the precision of numeric columns, and the number of bytes of storage for all other data types
scale	Displays the number of digits after the decimal point for numeric data type columns and zero for all other data types
“default”	The default value for the data type
“check”	The CHECK condition for the data type

Example

Display information about the user-defined data type `country_t`:

```
sp_iqdatatype country_t

type_name    creator    nulls    width    scale    “default”    “check”
country_t    DBA        U       15        0         (NULL)       (NULL)
```

See also

- *sp_iqcolumn Procedure* on page 447
- *sp_iqconstraint Procedure* on page 455
- *sp_iqevent Procedure* on page 484
- *sp_iqhelp Procedure* on page 489
- *sp_iqindex and sp_iqindex_alt Procedures* on page 496
- *sp_iqpkeys Procedure* on page 536
- *sp_iqprocparm Procedure* on page 540
- *sp_iq_reset_identity Procedure* on page 549
- *sp_iqtable Procedure* on page 575
- *sp_iqview Procedure* on page 590

sp_iqdbsize Procedure

Displays the size of the current database.

Syntax

```
sp_iqdbsize ( [ main ] )
```

Applies to
Simplex and multiplex.

Privileges
Requires the ALTER DATABASE system privilege. Users without the ALTER DATABASE system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description
Returns the total size of the database. Also returns the number of pages required to hold the database in memory and the number of IQ pages when the database is compressed (on disk).

If run on a multiplex database, the default parameter is **main**, which returns the size of the shared IQ store.

If run when there are no rows in any RLV-enabled tables, the Physical Blocks, the RLVLogBlocks and RLVLogKBytes columns will contain non-zero entries, and the remaining columns contain zeros. This indicate no row-level versioned tables.

Column Name	Description
Database	The path name of the database file.
Physical Blocks	Total database size in blocks. An IQ database consists of one or more dbspaces. Each dbspace has a fixed size, which is originally specified in units of megabytes. This megabyte quantity is converted to blocks using the IQ page size and the corresponding block size for that IQ page size. The Physical Blocks column reflects the cumulative total of each SAP Sybase IQ dbspace size, represented in blocks.
KBytes	The total size of the database in kilobytes. This value is the total size of the database in blocks (Physical Blocks in the previous **sp_iqdb-size** column) multiplied by the block size. The block size depends on the IQ page size.
Pages	The total number of IQ pages necessary to represent in memory all of the data stored in tables and the metadata for these objects. This value is always greater than or equal to the value of Compressed Pages (the next **sp_iqdbsize** column).
Compressed Pages	The total number of IQ pages necessary to store on disk the data in tables and metadata for these objects. This value is always less than or equal to the value of Pages (the previous **sp_iqdbsize** column), because SAP Sybase IQ compresses pages when the IQ page is written from memory to disk. The **sp_iqdbsize** Compressed Pages column represents the number of compressed pages.

Column Name	Description
NBlocks	The total size in blocks used to store the data in tables. This value is always less than or equal to the **sp_iqdbsize** Physical Blocks value.
Catalog Blocks	The total size in blocks used to store the metadata for tables.
RLVLogBlocks	Number of blocks used for log information for the RLV store.
RLVLogKBytes	Total size of the RLV log, in Kb.

Example

Displays size information for the database `iqdemo`:

```
sp_iqdbsize
```

```
Database

PhysicalBlocks KBytes Pages CompressedPages NBlocks CatalogBlocks
RLVLogBlocks RLVLogKBytes
============== ======= ===== =============== ======= =============
============ ============
/system1/sybase/IQ-16_0/demo/iqdemo.db
          1280    522   688            257    1119             18
```

sp_iqdbspace Procedure

Displays detailed information about each IQ dbspace.

Syntax

```
sp_iqdbspace [ dbspace-name ]
```

Applies to

Simplex and multiplex.

Privileges

Requires MANAGE ANY DBSPACE system privilege. Users without MANAGE ANY DBSPACE system privilege must be granted `EXECUTE` permission.

Description

Use the information from **sp_iqdbspace** to determine whether data must be moved, and for data that has been moved, whether the old versions have been deallocated.

Column Name	Description
DBSpaceName	Name of the dbspace as specified in the **CREATE DBSPACE** statement. Dbspace names are always case-insensitive, regardless of the **CREATE DATABASE...CASE IGNORE** or **CASE RESPECT** specification.

Column Name	Description
DBSpaceType	Type of the dbspace (`MAIN`, `SHARED_TEMP`, `TEMPORARY`, or `RLV`).
Writable	T (writable) or F (not writable).
Online	T (online) or F (offline).
Usage	Percent of dbspace currently in use by all files in the dbspace.
TotalSize	Total size of all files in the dbspace in the units B (bytes), K (kilobytes), M (megabytes), G (gigabytes), T (terabytes), or P (petabytes).
Reserve	Total reserved space that can be added to all files in the dbspace.
NumFiles	Number of files in the dbspace.
NumRWFiles	Number of read/write files in the dbspace.
Stripingon	F (Off).
StripeSize	Always 1, if disk striping is on.
BlkTypes	Space used by both user data and internal system structures.
OkToDrop	"Y" indicates the dbspace can be dropped; otherwise "N".

Values of the BlkTypes block type identifiers:

Identifier	Block Type
A	Active version
B	Backup structures
C	Checkpoint log
D	Database identity
F	Free list
G	Global free list manager
H	Header blocks of the free list
I	Index advice storage
M	Multiplex CM*
O	Old version
R	RLV free list manager
T	Table use

Identifier	Block Type
U	Index use
N	Column use
X	Drop at checkpoint

*The multiplex commit identity block (actually 128 blocks) exists in all IQ databases, even though it is not used by simplex databases.

Example

Displays information about dbspaces:

```
sp_iqdbspace;
```

Note: The following example shows objects in the `iqdemo` database to better illustrate output. `iqdemo` includes a sample user dbspace named `iq_main` that may not be present in your own databases.

DBSpaceName	DBSpaceType	Writable	Online	Usage	Total Size	Reserve	NumFiles	NumRWFiles	Stripingon	StripeSize	BlkTypes	OkToDrop
IQ_MAIN	MAIN	T	T	55	75M	200M	1	1	T	1K	1H, 5169A, 190	N
IQ__ SYSTEM_MAIN	MAIN	T	T	21	300M	50M	1	1	F	8K	1H, 7648F, 32D, 128M	N
IQ_SYSTEM_TEMP	TEMPORARY	T	T	1	100M	50M	1	1	F	8K	1H, 64F, 32A	N

See also

- *sp_iqindexinfo Procedure* on page 503
- *sp_iqdbspaceinfo Procedure* on page 470
- *sp_iqspaceinfo Procedure* on page 560

sp_iqdbspaceinfo Procedure

Displays the size of each object and subobject used in the specified table. Not supported for RLV dbspaces.

Syntax

```
sp_iqdbspaceinfo [ dbspace-name ] [ , owner_name ] [ ,
object_name ] [ , object-type  ]
```

Applies to

Simplex and multiplex.

Privileges

Requires the BACKUP DATABASE, SERVER OPERATOR, or MANAGE ANY DBSPACE system privileges. Users without one of these system privileges must be granted `EXECUTE` permission.

Usage

Parameter	Description
dbspace_name	If specified, **sp_iqdbspaceinfo** displays one line for each table that has any component in the specified dbspace. Otherwise, the procedure shows information for all dbspaces in the database.
owner_name	Owner of the object. If specified, **sp_iqdbspaceinfo** displays output only for tables with the specified owner. If not specified, **sp_iqdbspaceinfo** displays information on tables for all users in the database.
object_name	Name of the table. If not specified, **sp_iqdbspaceinfo** displays information on all tables in the database.
object_type	Valid **table** objects.

All parameters are optional, and any parameter may be supplied independent of another parameter’s value.

The **sp_iqdbspaceinfo** stored procedure supports wildcard characters for interpreting *dbspace_name*, *object_name*, and *owner_name*. It shows information for all dbspaces that match the given pattern in the same way the **LIKE** clause matches patterns inside queries.

Description

The procedure returns no results if you specify an RLV dbspace.

sp_iqdbspaceinfo shows the DBA the amount of space used by objects that reside on each dbspace. The DBA can use this information to determine which objects must be relocated before a dbspace can be dropped. The subobject columns display sizes reported in integer quantities followed by the suffix B, K, M, G, T, or P, representing bytes, kilobytes, megabytes, gigabytes, terabytes, and petabytes, respectively.

For tables, **sp_iqdbspaceinfo** displays subobject sizing information for all subobjects (using integer quantities with the suffix B, K, M, G, T, or P) sorted by *dbspace_name*, *object_name*, and *owner_name*.

Table 83. sp_iqdbspaceinfo Columns

Column Name	Description
dbspace_name	Name of the dbspace.
object_type	Type of the object (**table** or **joinindex** only).
owner	Name of the owner of the object.
object_name	Name of the object on the dbspace.
object_id	Global object ID of the object.
id	Table id of the object.
columns	Size of column storage space on the given dbspace.
indexes	Size of index storage space on the given dbspace. Does not use system-generated indexes (for example, HG indexes in unique constraints or FP indexes).
metadata	Size of storage space for metadata objects on the given dbspace.
primary_key	Size of storage space for primary key related objects on the given dbspace.
unique_constraint	Size of storage space for unique constraint-related objects on the given dbspace.
foreign_key	Size of storage space for foreign-key-related objects on the given dbspace.
dbspace_online	Indicates if the dbspace is online (**Y**) or offline (**N**).

If you run sp_iqdbspaceinfo against a server you have started with the -r switch (read-only), you see the error `Msg 13768, Level 14, State 0: SQL Anywhere Error -757: Modifications not permitted for read-only database`. This behavior is expected. The error does not occur on other stored procedures such as sp_iqdbspace, sp_iqfile, sp_iqdbspaceobjectinfo or sp_iqobjectinfo.

Examples

Note: These examples show objects in the iqdemo database to better illustrate output. iqdemo includes a sample user dbspace named iq_main that may not be present in your own databases.

Displays the size of all objects and subobjects in all tables in all dbspaces in the database:

```
sp_iqdbspaceinfo
```

```
dbspace_name    object_type  owner   object_name     object_id  id
  columns
iq_main        table       DBA     emp1            3689     741   96K
iq_main        table       DBA     iq_dummy        3686     740   24K
iq_main        table       DBA     sale            3698     742   96K
iq_main         table        GROUPO  Contacts         3538     732
   288K
iq_main         table        GROUPO  Customers        3515     731
   240K
iq_main        table       GROUPO  Departments      3632     738   72K
iq_main         table        GROUPO  Employees        3641     739
   408K
iq_main         table        GROUPO  FinancialCodes    3612     736
   72K
iq_main        table       GROUPO  FinancialData    3621     737   96K
iq_main         table        GROUPO  Products
 3593     735   272K
iq_main         table        GROUPO  SalesOrderItems   3580     734
   120K
iq_main         table        GROUPO  SalesOrders       3565     733
   144K

indexes  metadata  primary_key  unique_constraint  foreign_key  dbsp
ace_online
0B       1.37M     0B           0B                 0B           Y
0B       464K      0B           0B                 0B           Y
0B       1.22M     0B           0B                 0B           Y
0B       5.45M     24K          0B                 48K          Y
48K      4.63M     24K          0B                 0B           Y
0B       1.78M     24K          0B                 48K          Y
0B       8.03M     24K          0B                 48K          Y
0B       1.53M     24K          0B                 0B           Y
0B       2.19M     24K          0B                 48K          Y
192K     4.67M     24K          0B                 0B           Y
0B       2.7M      24K          0B                 104K         Y
0B       3.35M     24K          0B                 144K         Y
```

Displays the size of all objects and subobjects owned by a specified user in a specified dbspace in the database:

```
sp_iqdbspaceinfo iq_main,GROUPO
```

```
dbspace_name    object_type  owner   object_name     object_id  id
  columns
iq_main         table        GROUPO  Contacts          3538     732
   288K
iq_main         table        GROUPO  Customers         3515     731
   240K
iq_main        table       GROUPO  Departments      3632    738   72K
iq_main         table        GROUPO  Employees         3641     739
   408K
iq_main         table        GROUPO  FinancialCodes    3612     736
   72K
iq_main        table       GROUPO  FinancialData    3621    737   96K
iq_main         table        GROUPO  Products          3593     735
   272K
iq_main         table        GROUPO  SalesOrderItems   3580     734
   120K
iq_main         table        GROUPO  SalesOrders       3565     733
   144K

indexes  metadata  primary_key  unique_constraint  foreign_key  dbsp
ace_online
0B       5.45M     24K          0B                 48K          Y
48K      4.63M     24K          0B                 0B           Y
0B       1.78M     24K          0B                 48K          Y
0B       8.03M     24K          0B                 48K          Y
0B       1.53M     24K          0B                 0B           Y
0B       2.19M     24K          0B                 48K          Y
192K     4.67M     24K          0B                 0B           Y
0B       2.7M      24K          0B                 104K         Y
0B       3.35M     24K          0B                 144K         Y
```

Displays the size of a specified object and its subobjects owned by a specified user in a specified dbspace in the database:

```
sp_iqdbspaceinfo iq_main,GROUPO,Departments
```

```
dbspace_name    object_type  owner   object_name     object_id  id
  columns
iq_main        table       GROUPO  Departments      3632    738   72K

indexes  metadata  primary_key  unique_constraint  foreign_key  dbsp
ace_online
0B       1.78M     24K          0B                 48K          Y
```

See also

- *sp_iqindexinfo Procedure* on page 503
- *sp_iqdbspace Procedure* on page 467
- *sp_iqspaceinfo Procedure* on page 560

sp_iqdbspaceobjectinfo Procedure

Lists objects and subobjects of type table (including columns, indexes, metadata, primary keys, unique constraints, foreign keys, and partitions) for a given dbspace. Not supported for RLV dbspaces.

Syntax

```
sp_iqdbspaceobjectinfo [ dbspace-name ] [ , owner_name ] [ ,
object_name ] [ , object-type ]
```

Privileges

No specific system privilege required.

Usage

All parameters are optional and any parameter may be supplied independent of the value of other parameters.

Table 84. Parameters

Parameter	Description
dbspace-name	If specified, **sp_iqdbspaceobjectinfo** displays output only for the specified dbspace. Otherwise, it shows information for all dbspaces in the database.
owner-name	Owner of the object. If specified, **sp_iqdbspaceobjectinfo** displays output only for tables with the specified owner. If not specified, **sp_iqdbspaceobjectinfo** displays information for tables for all users in the database.
object-name	Name of the table. If not specified, **sp_iqdbspaceobjectinfo** displays information for all tables in the database.
object-type	Valid object types for **table** objects.

The **sp_iqdbspaceobjectinfo** stored procedure supports wildcard characters for interpreting *dbspace_name*, *object_name*, and *owner_name*. It displays information for all dbspaces that match the given pattern in the same way as the **LIKE** clause matches patterns inside queries.

Description

The procedure returns no results if you specify an RLV dbspace.

For tables, **sp_iqdbspaceobjectinfo** displays summary information for all associated subobjects sorted by dbspace_name, owner and object_name.

sp_iqdbspaceobjectinfo displays the following information, based on the input parameter values:

Table 85. sp_iqdbspaceobjectinfo columns

Column Name	Description
dbspace_name	Name of the dbspace.
dbspace_id	Identifier of the dbspace.
object_type	Table.
owner	Name of the owner of the object.
object_name	Name of the table object on the dbspace.
object_id	Global object ID of the object.
id	Table ID of the object.
columns	Number of table columns which are located on the given dbspace. If a column or one of the column-partitions is located on a dbspace, it is counted to be present on that dbspace. The result is shown in the form n/N (n out of total N columns of the table are on the given dbspace).
indexes	Number of user-defined indexes on the table which are located on the given dbspace. Shown in the form n/N (n out of total N indexes on the table are on the given dbspace). This does not contain indexes which are system-generated, such as FP indexes and HG indexes in the case of unique constraints.
metadata	Boolean field (Y/N) that denotes whether the metadata information of the subobject is also located on this dbspace.
primary_key	Boolean field (1/0) that denotes whether the primary key of the table, if any, is located on this dbspace.
unique_constraint	Number of unique constraints on the table that are located on the given dbspace. Appears in the form n/N (n out of total N unique constraints on the table are in the given dbspace).
foreign_key	Number of foreign_keys on the table that are located on the given dbspace. Appears in the form n/N (n out of total N foreign keys on the table are in the given dbspace).
partitions	Number of partitions of the table that are located on the given dbspace. Appears in the form n/N (n out of total N partitions of the table are in the given dbspace).

Examples

These examples show objects in the `iqdemo` database to better illustrate output. `iqdemo` includes a sample user dbspace named `iq_main` that may not be present in your own databases.

Displays information about a specific dbspace in the database:

```
sp_iqdbspaceobjectinfo iq_main
```

```
dbspace_name  dbspace_id object_type  owner   object_name  object_i
d  id  columns
iq_main       16387      table        DBA     emp1           3689
   741  4/4
iq_main       16387      table        DBA     iq_dummy       3686
   740  1/1
iq_main       16387      table        DBA     sale           3698
   742  4/4
iq_main       16387      table        GROUPO  Contacts       3538
   732  12/12
iq_main       16387      table        GROUPO  Customers      3515
   731  10/10
iq_main       16387      table        GROUPO  Departments    3632
   738  3/3
iq_main       16387      table        GROUPO  Employees      3641
   739  21/21
iq_main       16387      table        GROUPO  FinancialCodes 3612
   736  3/3
iq_main       16387      table        GROUPO  FinancialData  3621
   737  4/4
iq_main       16387      table        GROUPO  Products       3593
   735  8/8
iq_main       16387      table        GROUPO  SalesOrderItems3580
   734  5/5
iq_main       16387      table        GROUPO  SalesOrders    3565
   733  6/6

indexes  metadata  primary_key  unique_constraint  foreign_key  part
itions
0/0      Y                   0  0/0                0/0          0/0
0/0      Y                   0  0/0                0/0          0/0
0/0      Y                   0  0/0                0/0          0/0
0/0      Y                   1  0/0                1/1          0/0
1/1      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                1/1          0/0
0/0      Y                   1  0/0                1/1          0/0
0/0      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                1/1          0/0
4/4      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                2/2          0/0
0/0      Y                   1  0/0                3/3          0/0
```

Displays information about the objects owned by a specific user in a specific dbspace in the database:

```
sp_iqdbspaceobjectinfo iq_main,GROUPO
```

```
dbspace_name  dbspace_id object_type  owner   object_name    object_
id  id  columns
iq_main       16387      table        GROUPO  Contacts       3538
     732 2/12
iq_main       16387      table        GROUPO  Customers      3515
     731 10/10
iq_main       16387      table        GROUPO  Departments    3632
     738 3/3
iq_main       16387      table        GROUPO  Employees      3641
     739 21/21
iq_main       16387      table        GROUPO  FinancialCodes 3612
     736 3/3
iq_main       16387      table        GROUPO  FinancialData  3621
     737 4/4
iq_main       16387      table        GROUPO  Products       3593
     735 8/8
iq_main       16387      table        GROUPO  SalesOrderItems3580
     734 5/5
iq_main       16387      table        GROUPO  SalesOrders    3565
     733 6/6

indexes  metadata  primary_key  unique_constraint  foreign_key  part
itions
0/0      Y                   1  0/0                1/1          0/0
1/1      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                1/1          0/0
0/0      Y                   1  0/0                1/1          0/0
0/0      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                1/1          0/0
4/4      Y                   1  0/0                0/0          0/0
0/0      Y                   1  0/0                2/2          0/0
0/0      Y                   1  0/0                3/3          0/0
```

In this example, the commands move all tables on `dbspace_x` to `dbspace_y`.

```
SELECT 'ALTER TABLE ' || owner || '.' ||
object_name || ' MOVE TO dbspace_y;'
FROM sp_iqdbspaceobjectinfo()
WHERE object_type = 'table' AND
dbspace_name = 'dbspace_x';
```

The following **ALTER TABLE** commands are the result:

```
ALTER TABLE DBA.dt1 MOVE TO dbspace_y;
ALTER TABLE DBA.dt2 MOVE TO dbspace_y;
ALTER TABLE DBA.dt3 MOVE TO dbspace_y;
```

sp_iqdbstatistics Procedure

Reports results of the most recent **sp_iqcheckdb**.

Syntax

```
sp_iqdbstatistics
```

Privileges

Requires the ALTER DATABASE system privilege. Users without the ALTER DATABASE system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

Displays the database statistics collected by the most recent execution of **sp_iqcheckdb**.

Example

The following example shows the output from **sp_iqdbstatistics**. For this example, the most recent execution of **sp_iqcheckdb** was the command **sp_iqcheckdb 'allocation database'**.

```
           DB Statistics                     Value               Flags
=====================================|============================|
=====
DBCC Allocation Mode Report          |                            |
=====================================|============================|
=====
** DBCC Status                     |Errors Detected            |*****
   DBCC Work units Dispatched        |163                         |
   DBCC Work units Completed         |163                         |
=====================================|============================|
=====
Allocation Summary                   |                            |
=====================================|============================|
=====
   Blocks Total                      |8192                        |
   Blocks in Current Version         |4954                        |
   Blocks in All Versions            |4954                        |
   Blocks in Use                     |4986                        |
   % Blocks in Use                   |60                          |
** Blocks Leaked                   |32                         |*****
                                     |                            |
=====================================|============================|
=====
Allocation Statistics                |                            |
=====================================|============================|
=====
   Blocks Created in Current TXN     |382                         |
   Blocks To Drop in Current TXN     |382                         |
   Marked Logical Blocks             |8064                        |
   Marked Physical Blocks            |4954                        |
   Marked Pages                      |504                         |
   Blocks in Freelist                |126553                      |
   Imaginary Blocks                  |121567                      |
   Highest PBN in Use                |5432                        |
** 1st Unowned PBN                 |452                        |*****
   Total Free Blocks                 |3206                        |
   Usable Free Blocks                |3125                        |
   % Free Space Fragmented           |2                           |
   Max Blocks Per Page               |16                          |
   1  Block Page Count               |97                          |
   3  Block Page Count               |153                         |
   4  Block Page Count                    |14                         |
   ...
   9  Block Hole Count                    |2                          |
   16 Block Hole Count                    |194                        |
                                          |                           |
   Database Objects Checked               |1                          |
   B-Array Count                          |1                          |
   Blockmap Identity Count                |1                          |
=====================================|===========================|
=====
Connection Statistics                     |                           |
=====================================|===========================|
=====
```

sp_iqdroplogin Procedure

Drops an SAP Sybase IQ user account.

Syntax1

call sp_iqdroplogin (‘*userid*’)

Syntax2

sp_iqdroplogin ‘*userid*’

Syntax3

sp_iqdroplogin *userid*

Syntax4

sp_iqdroplogin (‘*userid*’)

Privileges

Requires the MANAGE ANY USER system privilege. Users without MANAGE ANY USER system privilege must be granted `EXECUTE` permission.

Usage

Table 86. Parameters

Parameter	Description
userid	ID of the user to drop.

Description

sp_iqdroplogin drops the specified user.

Examples

These commands all remove the user `rose`:

```
sp_iqdroplogin 'rose'
sp_iqdroplogin rose
```

```
call sp_iqdroplogin ('rose')
```

See also

- *sp_iqaddlogin Procedure* on page 429

sp_iqemptyfile Procedure

Empties a dbfile and moves the objects in the dbfile to another available read-write dbfile in the same dbspace. Not available for files in an RLV dbspace.

Syntax

```
sp_iqemptyfile ( logical-file--name )
```

Privileges

Requires at least one system privilege from each group:

Group 1	Group 2
BACKUP DATABASE	INSERT ANY TABLE
SERVER OPERATOR	UPDATE ANY TABLE
ALTER DATABASE	DELETE ANY TABLE
	ALTER ANY TABLE
	LOAD ANY TABLE
	TRUNCATE ANY TABLE
	ALTER ANY OBJECT

Users without the required system privileges must be granted `EXECUTE` permission to run the stored procedure.

Description

sp_iqemptyfile empties a dbfile. The dbspace must be read-only before you can execute the **sp_iqemptyfile** procedure. The procedure moves the objects in the file to another available read-write dbfile in the same dbspace. If there is no other read-write dbfile available, then SAP Sybase IQ displays an error message.

Note: In a multiplex environment, you can run **sp_iqemptyfile** only on the coordinator. There must be one read-write dbspace available for the procedure to succeed.

If the dbfile is in an RLV dbspace, then this error message displays:

```
Cannot empty files in an rlv store dbspace.
```

Example

Empties dbfile **dbfile1**:

```
sp_iqemptyfile 'dbfile1'
```

sp_iqestdbspaces Procedure

Estimates the number and size of dbspaces needed for a given total index size.

Syntax

sp_iqestdbspaces (*db_size_in_bytes*, *iq_page_size*, *min_#_of_bytes*, *max_#_of_bytes*)

Privileges

Requires the MANAGE ANY DBSPACE or ALTER DATABASE system privileges. Users without one of these system privileges must be granted `EXECUTE` permission.

Description

Displays information about the number and size of dbspace segments based on the size of the database, the IQ page size, and the range of bytes per dbspace segment. This procedure assumes that the database was created with the default block size for the specified IQ page size; otherwise, the returned estimated values are incorrect.

Table 87. sp_iqestdbspaces Parameters

Name	Datatype	Description
db_size_in_bytes	`decimal(16)`	Size of the database in bytes.
iq_page_size	`smallint`	The page size defined for the IQ segment of the database (must be a power of 2 between 65536 and 524288; the default is 131072).
min_#_of_bytes	`int`	The minimum number of bytes per dbspace segment. The default is 20,000,000 (20MB).
max_#_of_bytes	`int`	The maximum number of bytes per dbspace segment. The default is 2,146,304,000 (2.146GB).

Usage

sp_iqestdbspaces reports several recommendations, depending on how much of the data is unique:

Recommendation	Description
min	If there is little variation in data, you can choose to create only the dbspace segments of the sizes recommended as **min**. These recommendations reflect the best possible compression on data with the least possible variation.
avg	If your data has an average amount of variation, create the dbspace segments recommended as **min**, plus additional segments of the sizes recommended as **avg**.
max	If your data has a high degree of variation (many unique values), create the dbspace segments recommended as **min**, **avg**, and **max**.
spare	If you are uncertain about the number of unique values in your data, create the dbspace segments recommended as **min**, **avg**, **max**, and **spare**. You can always delete unused segments after loading your data, but creating too few can cost you some time.

sp_iqestdbspaces Procedure Example

Use the example as a reference for **sp_iqestdbspaces** usage.

```
sp_iqestdbspaces 12000000000, 65536, 500000000, 2146304000
```

dbspace files	Type	Size	Msg
1	min	2146304000	
2	min	2146304000	
3	min	507392000	
4	avg	2146304000	
5	max	2053697536	
6	spare	1200001024	

This example estimates the size and number of dbspace segments needed for a 12GB database. SAP Sybase IQ recommends that you create a minimum of 3 segments (listed as **min**) for the best compression, if you expect little uniqueness in the data. If the data has an average amount of variation, 1 more segment (listed as **avg**) should be created. Data with a lot of variation (many unique values, requiring extensive indexing), may require 1 more segment (listed as **max**). You can ensure that your initial load succeeds by creating a spare segment of

1200001024 bytes. Once you have loaded the database, you can delete any unused dbspace segments.

Using sp_iqestdbspaces With Other System Stored Procedures

You need to run two stored procedures to provide the *db_size_in_bytes* parameter needed by **sp_iqestdbspaces**.

Results of **sp_iqestdbspaces** are only estimates, based on the average size of an index. The actual size depends on the data stored in the tables, particularly on how much variation there is in the data.

Sybase strongly recommends that you create the spare dbspace segments, because you can delete them later if they are unused.

1. Run **sp_iqestjoin** for all the table pairs you expect to join frequently.
2. Select one of the suggested index sizes for each pair of tables.
3. Total the index sizes you selected for all tables.
4. Run **sp_iqestspace** for all tables.
5. Total all of the RAW DATA index sizes returned by **sp_iqestspace**.
6. Add the total from step 3 to the total from step 5 to determine total index size.
7. Use the total index size calculated in step 6 as the *db_size_in_bytes* parameter in **sp_iqestdbspaces**.

sp_iqestspace Procedure

Estimates the amount of space needed to create an index based on the number of rows in the table.

Syntax

```
sp_iqestspace ( table_name, #_of_rows, iq_page_size )
```

Privileges

User must be a table owner or have the CREATE ANY INDEX, ALTER ANY INDEX, CREATE ANY OBJECT, or ALTER ANY OBJECT system privileges. Users without one of these system privileges must be granted `EXECUTE` permission to run the stored procedure.

Description

Displays the amount of space that a database requires based on the number of rows in the underlying database tables and on the database IQ page size. This procedure assumes that the database was created with the default block size for the specified IQ page size (or else the estimate is incorrect). This table lists the **sp_iqestspace** parameters.

Table 88. sp_iqestspace parameters

Name	Datatype	Description
table_name	`char(256)`	Name of the table
#_of_rows	`int`	Number of rows in the table
iq_page_size	`smallint`	The page size defined for the IQ segment of the database (must be a power of 2 between 65536 and 524288; the default is 131072)

sp_iqevent Procedure

Displays information about system and user-defined events.

Syntax

```
sp_iqevent [ event-name ], [ event-owner ], [ event-type ]
```

Privileges

No specific system privileges are required to run this procedure.

Usage

Table 89. Parameter

Parameter	Description
event-name	The name of the event.
event-owner	The owner of the event.
event-type	The type of event. Allowed values are: • **SYSTEM**: displays information about system events (events owned by user `SYS` or `dbo`) only • **ALL**: displays information about user and system events • Any other value: displays information about user events

The **sp_iqevent** procedure can be invoked without any parameters. If no parameters are specified, only information about user events (events not owned by `dbo` or `SYS`) is displayed by default.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqevent NULL, NULL, SYSTEM` and `sp_iqevent NULL, user1`.

Table 90. sp_iqevent usage examples

Syntax	Output
sp_iqevent	Displays information about all user events in the database
sp_iqevent e1	Displays information about the event `e1`
sp_iqevent non_existing_event	No rows returned, as the event `non_existing_event` does not exist
sp_iqevent NULL, DBA	Displays information about all events owned by `DBA`
sp_iqevent e1, DBA	Displays information about the event `e1` owned by `DBA`
sp_iqevent ev_iqbegintxn	`ev_iqbegintxn` is a system-defined event. If there is no user-defined event also named `ev_iqbegintxn`, no rows are returned. (By default only user-defined events are returned.)
sp_iqevent ev_iqbegintxn, dbo	No rows returned, as the event `ev_iqbegintxn` is not a user event (by default only user events returned)
sp_iqevent NULL, NULL, SYSTEM	Displays information about all system events (owned by `dbo` or `SYS`)
sp_iqevent ev_iqbegintxn, NULL, SYSTEM	Displays information about the system event `ev_iqbegintxn`
sp_iqevent ev_iqbegintxn, dbo, ALL	Displays information about the system event `ev_iqbegintxn` owned by `dbo`

Description

The **sp_iqevent** stored event displays information about events in a database. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if *event-name* is specified, only information about the specified event is displayed. If *event-owner* is specified, **sp_iqevent** only returns information about events owned by the specified owner. If no parameters are specified, **sp_iqevent** displays information about all the user events in the database.

The **sp_iqevent** procedure returns information in the following columns:

Table 91. sp_iqevent columns

Column name	Description
event_name	The name of the event
event_owner	The owner of the event

Column name	Description
event_type	For system events, the event type as listed in the SYSEVENTTYPE system table
enabled	Indicates whether or not the event is allowed to fire (Y/N)
action	The event handler definition
condition	The **WHERE** condition used to control firing of the event handler
location	The location where the event is allowed to fire: • C = consolidated • R = remote • A = all
remarks	A comment string

Examples

Display information about the user-defined event `e1`:

```
sp_iqevent e1

event_name    event_owner    event_type    enabled    action
e1            DBA           (NULL)         Y          (NULL)

condition    location    remarks
(NULL)       A           (NULL)
```

Display information about all system events:

```
sp_iqevent NULL, NULL, SYSTEM

event_name      event_owner    event_type      enabled   action
ev_iqbegintxn   dbo            IQTLVAvailable   Y         begin call
                                                     dbo.sp_iqlog...
ev_iqmpxcompact dbo          (NULL)          N         begin Declare
                                                         _Catalog...

condition    location    remarks
(NULL)       A           (NULL)
(NULL)       A           (NULL)
```

See also

- *sp_iqcolumn Procedure* on page 447
- *sp_iqconstraint Procedure* on page 455
- *sp_iqdatatype Procedure* on page 463
- *sp_iqhelp Procedure* on page 489
- *sp_iqindex and sp_iqindex_alt Procedures* on page 496
- *sp_iqpkeys Procedure* on page 536

- *sp_iqprocparm Procedure* on page 540
- *sp_iq_reset_identity Procedure* on page 549
- *sp_iqtable Procedure* on page 575
- *sp_iqview Procedure* on page 590

sp_iqfile Procedure

Displays detailed information about each dbfile in a dbspace.

Syntax

```
sp_iqfile [ dbspace-name ]
```

Applies to

Simplex and multiplex.

Privileges

Requires the MANAGE ANY DBSPACE system privilege. Users without the MANAGE ANY DBSPACE system privilege must be granted `EXECUTE` permission.

Description

sp_iqfile displays the usage, properties, and types of data in each dbfile in a dbspace. You can use this information to determine whether data must be moved, and for data that has been moved, whether the old versions have been deallocated.

Column Name	Description
DBSpaceName	Name of the dbspace as specified in the **CREATE DBSPACE** statement. Dbspace names are always case-insensitive, regardless of the **CREATE DATABASE**...**CASE IGNORE** or **CASE RESPECT** specification.
DBFileName	Logical file name.
Path	Location of the physical file or raw partition.
SegmentType	Type of dbspace (`MAIN`, `TEMPORARY`, or `RLV`).
RWMode	Mode of the dbspace: always read-write (RW).
Online	T (online) or F (offline).
Usage	Percent of dbspace currently in use by this file in the dbspace. When run against a secondary node in a multiplex configuration, this column displays NA.
DBFileSize	Current size of the file or raw partition. For a raw partition, this size value can be less than the physical size.
Reserve	Reserved space that can be added to this file in the dbspace.

Column Name	Description
StripeSize	Always 1, if disk striping is on.
BlkTypes	Space used by both user data and internal system structures.
FirstBlk	First IQ block number assigned to the file.
LastBlk	Last IQ block number assigned to the file.
OkToDrop	"Y" indicates the file can be dropped; otherwise "N".

Identifier	Block Type
A	Active Version
B	Backup Structures
C	Checkpoint Log
D	Database Identity
F	Free list
G	Global Free list Manager
H	Header Blocks of the Free List
I	Index Advice Storage
M	Multiplex CM*
O	Old Version
R	RLV Free list manager
T	Table Use
U	Index Use
N	Column Use
X	Drop at Checkpoint

*The multiplex commit identity block (actually 128 blocks) exists in all IQ databases, even though it is not used by simplex databases.

Example

Displays information about the files in the dbspaces:

```
sp_iqfile;
```

```

sp_iqfile;
```

```
DBSpaceName,DBFileName,Path,SegmentType,RWMode,Online,
Usage,DBFileSize,Reserve,StripeSize,BlkTypes,FirstBlk,
LastBlk,OkToDrop

'IQ_SYSTEM_MAIN','IQ_SYSTEM_MAIN','/sun1-c1/users/smith/mpx/m/
mpx_db.iq','MAIN','RW','T','21','
2.92G','0B','1K','1H,76768F,32D,19A,185O,128M,34B,32C'
,1,384000,'N'

'mpx_main1','mpx_main1','/sun1-c1/users/smith/mpx/m/
mpx_main1.iq','MAIN','RW','T','1'
,'100M','0B','1K','1H',1045440,1058239,'N'

'IQ_SHARED_TEMP','sharedfile1_bcp','/sun1-c1/users/smith/mpx/m/
f1','SHARED_TEMP','RO','T','0',
'50M','0B','1K','1H',1,6400,'N'

'IQ_SHARED_TEMP','sharedfile2_bcp','/sun1-c1/users/smith/mpx/m/
f2','SHARED_TEMP','RO','T','0',
'50M','0B','1K','1H',1045440,1051839,'N'

'IQ_SYSTEM_TEMP','IQ_SYSTEM_TEMP','/sun1-c1/users/smithmpx/m/
mpx_db.iqtmp','TEMPORARY','RW',
'T','1','2.92G','0B','1K','1H,64F,33A',1,384000,'N'
```

sp_iqhelp Procedure

Displays information about system and user-defined objects and data types.

Syntax

sp_iqhelp [*obj-name*], [*obj-owner*], [*obj-category*], [*obj-type*]

Privileges

No specific system privilege is required to run this procedure.

Usage

Table 92. Parameters

Parameter	Description
obj-name	The name of the object.
obj-owner	The owner of the object.

Parameter	Description
obj-category	An optional parameter that specifies the category of the object. Columns, constraints, and indexes are associated with tables and cannot be queried directly. When a table is queried, the information about columns, indexes, and constraints associated with that table is displayed. If the specified object category is not one of the allowed values, an `“Invalid object category”` error is returned.
obj-type	The type of object. Allowed values are: • **SYSTEM**: displays information about system objects (objects owned by user `SYS` or `dbo`) only • **ALL**: displays information about all objects By default, only information about non-system objects is displayed. If the specified object type is not **SYSTEM** or **ALL**, an `“Invalid object type”` error is returned.

Table 93. sp_iqhelp obj-category parameter values

object-type **parameter**	Specifies
“table”	The object is a base table
“view”	The object is an view
“procedure”	The object is a stored procedure or function
“event”	The object is an event
“datatype”	The object is a system or user-defined data type

The **sp_iqhelp** procedure can be invoked without any parameters. If no parameters are specified, **sp_iqhelp** displays information about all independent objects in the database, that is, base tables, views, stored procedures, functions, events, and data types.

If you do not specify any of the first three parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqhelp NULL, NULL, NULL, SYSTEM` and `sp_iqhelp NULL, user1, “table”`.

Enclose the *obj-category* parameter in single or double quotes., except when NULL.

If **sp_iqhelp** does not find an object in the database that satisfies the specified description, the error "`No object found for the given description`" is returned.

Table 94. sp_iqhelp usage examples

Syntax	Output
sp_iqhelp	Displays summary information about all user-defined tables, views, procedures, events, and data types in the database
sp_iqhelp t1, u1, "table"	Displays information about table `t1` owned by user `u1` and the columns, indexes, and constraints associated with `t1`
sp_iqhelp NULL, u1, "view"	Displays information about view `v1` owned by user `u1` and the columns associated with `v1`
sp_iqhelp sp2	Displays information about the procedure `sp2` and the parameters of `sp2`
sp_iqhelp e1	Displays information about the event `e1`
sp_iqhelp dt1	Displays information about the data type `dt1`
sp_iqhelp NULL, NULL, NULL, SYSTEM	Displays summary information about all system objects (owned by `dbo` or `SYS`)
sp_iqhelp non_existing_obj	Error "`Object 'non_existing_obj' not found`" returned, as the object **non_existing_obj** does not exist
sp_iqhelp NULL, non_existing_user	Error "`User 'non_existing_user' not found`" returned, as the user **non_existing_user** does not exist
sp_iqhelp t1, NULL, "apple"	Error "`Invalid object category 'apple'`" returned, as **"apple"** is not an allowed value for *obj-category*
sp_iqhelp t1, NULL, NULL, "USER"	Error "`Invalid object type 'USER'`" returned, as **"USER"** is not an allowed value for *obj-type*

Description

The **sp_iqhelp** stored procedure displays information about system and user-defined objects and data types in an IQ database. Objects supported by **sp_iqhelp** are tables, views, columns, indexes, constraints, stored procedures, functions, events, and data types.

If you specify one or more parameters, the result is filtered by the specified parameters. For example, if *obj-name* is specified, only information about the specified object is displayed. If *obj-owner* is specified, **sp_iqhelp** returns information only about objects owned by the specified owner. If no parameters are specified, **sp_iqhelp** displays summary information about all user-defined tables, views, procedures, events, and data types in the database.

The **sp_iqhelp** procedure returns either summary or detailed information, depending on whether the specified parameters match multiple objects or a single object. The output columns of **sp_iqhelp** are similar to the columns displayed by the stored procedures **sp_iqtable**, **sp_iqindex**, **sp_iqview**, and **sp_iqconstraint**.

When multiple objects match the specified **sp_iqhelp** parameters, **sp_iqhelp** displays summary information about those objects.

Table 95. sp_iqhelp summary information

Object type	Columns displayed
base table	table_name, table_owner, server_type, location, table_constraints, remarks
view	view_name, view_creator, view_def, server_type, location, remarks
stored procedure	proc_name, proc_creator, proc_defn, replicate, srvid, remarks
function	proc_name, proc_creator, proc_defn, replicate, remarks
event	event_name, event_creator, enabled, location, event_type, action, external_action, condition, remarks
system and user-defined data types	type_name, creator, nulls, width, scale, default, check

When a single object matches the specified **sp_iqhelp** parameters, **sp_iqhelp** displays detailed information about the object.

Table 96. sp_iqhelp detailed information

Object type	Description	Columns
table	Displays information about the specified base table, its columns, indexes, and constraints.	• Table columns: table_name, table_owner, server_type, location, table_constraints, remarks • Column columns: column_name, domain_name, width, scale, nulls, default, check, pkey, user_type, cardinality, est_cardinality, remarks • Index columns: index_name, column_name, index_type, unique_index, location, remarks • Constraint columns: constraint_name (role), column_name, index_name, constraint_type, foreigntable_name, foreigntable_owner, foreigncolumn_name, foreignindex_name, location
view	Displays information about the specified view and its columns	• View columns: view_name, view_creator, view_def, server_type, location, remarks • Column columns: column_name, domain_name, width, scale, nulls, default, check, pkey, user_type, cardinality, est_cardinality, remarks
stored procedure	Displays information about the specified procedure and its parameters	• Procedure columns: proc_name, proc_creator, proc_defn, replicate, srvid, remarks • Parameter columns: parameter_name, type, width, scale, default, mode
function	Displays information about the specified function and its parameters	• Function columns: proc_name, proc_creator, proc_defn, replicate, srvid, remarks • Parameter columns: parameter_name, type, width, scale, default, mode
event	Displays information about the specified event	• Event columns: event_name, event_creator, enabled, location, event_type, action, external_action, condition, remarks
data type	Displays information about the specified data type	• Data type columns: type_name, creator, nulls, width, scale, default, check

Note: Procedure definitions (proc-defn) of system procedures are encrypted and hidden from view.

For descriptions of the individual output columns, refer to the related stored procedure. For example, for a description of the table column, see the **sp_iqtable** procedure.

Examples

Display detailed information about the table `sale`:

```
sp_iqhelp sale
```

```

Table_name Table_owner Server_type Location dbspace_id isPartitioned
 table_constraints
==========  ===========  ==========  =======  ==
======= =============
sale      DBA         IQ           Main    16387      N

Remarks  table_constraints
=======  ================== (NULL)   (NULL)

column_name domain_name width scale nulls default cardinality
==========  =========== ===== ===== ===== ======= ===========
prod_id     integer     4     0     Y    (NULL)   0
month_num   integer     4     0     Y    (NULL)   0
rep_id      integer     4     0     Y    (NULL)   0
sales       integer     4     0     Y    (NULL)   0

  est_cardinality    isPartitioned    remarks    check
  ==============     =============    =======    =====
  0                  N               (NULL)      (NULL)
  0                  N               (NULL)      (NULL)
  0                  N               (NULL)      (NULL)
  0                  N               (NULL)      (NULL)

index_name           column_name  index_type  unique_index  location
==========           ===========  =========== ===========   ========
ASIQ_IDX_T463_C2_FP  month_num    FP          N             Main
ASIQ_IDX_T463_C1_FP  prod_id      FP          N             Main
ASIQ_IDX_T463_C3_FP  rep_id       FP          N             Main
ASIQ_IDX_T463_C4_FP  sales        FP          N             Main

  remarks
  =======
  (NULL)
  (NULL)
  (NULL)
  (NULL)
```

Display detailed information about the procedure `sp_customer_list`:

```
sp_iqhelp sp_customer_list
proc_name    proc_owner    proc_defn
==========  ===========    =========
sp_customer_list    DBA    create procedure DBA.sp_customer_list()
```

```
                          result(id integer company_name char(35))
                          begin
                          select id company_name from Customers
                          end

  replicate    srvid    remarks
  =========    =====    =======
  N            (NULL)   (NULL)

parm_name      parm_type  parm_mode  domain_name  width  scale
=========      =========  =========  ===========  =====  =====
id             result     out        integer      4      0
company_name   result     out        char         35     0

  default
  =======
  (NULL)
```

See also

- *sp_iqcolumn Procedure* on page 447
- *sp_iqconstraint Procedure* on page 455
- *sp_iqdatatype Procedure* on page 463
- *sp_iqevent Procedure* on page 484
- *sp_iqindex and sp_iqindex_alt Procedures* on page 496
- *sp_iqpkeys Procedure* on page 536
- *sp_iqprocparm Procedure* on page 540
- *sp_iq_reset_identity Procedure* on page 549
- *sp_iqtable Procedure* on page 575
- *sp_iqview Procedure* on page 590

sp_iqhelp Compatibility with Adaptive Server Enterprise

The SAP Sybase IQ **sp_iqhelp** stored procedure is similar to the Adaptive Server Enterprise **sp_help** procedure, which displays information about any database object listed in the `SYSOBJECTS` system table and about system and user-defined data types.

SAP Sybase IQ has some architectural differences from Adaptive Server Enterprise in terms of types of objects supported and the namespace of objects. In Adaptive Server Enterprise, all objects (tables, views, stored procedures, logs, rules, defaults, triggers, check constraints, referential constraints, and temporary objects) are stored in the `SYSOBJECTS` system table and are in the same namespace. The objects supported by SAP Sybase IQ (tables, views, stored procedures, events, primary keys, and unique, check, and referential constraints) are stored in different system tables and are in different namespaces. For example, in SAP Sybase IQ a table can have the same name as an event or a stored procedure.

Because of the architectural differences between SAP Sybase IQ and Adaptive Server Enterprise, the types of objects supported by and the syntax of SAP Sybase IQ **sp_iqhelp** are different from the supported objects and syntax of Adaptive Server Enterprise **sp_help**;

however, the type of information about database objects that is displayed by both stored procedures is similar.

sp_iqindex and sp_iqindex_alt Procedures

Lists information about indexes.

Syntax 1

```
sp_iqindex ( [ table_name ],[column_name ],[table_owner ] )
```

Syntax 2

```
sp_iqindex [table_name='tablename' ],
[column_name='columnname' ],[table_owner='tableowner' ]
```

Syntax 3

```
sp_iqindex_alt ( [ table_name ],[column_name ],[table_owner ] )
```

Syntax 4

```
sp_iqindex_alt [table_name='tablename' ],
[column_name='columnname' ],[table_owner='tableowner' ]
```

Privileges

No specific system privilege is required to run these procedures.

Usage

Table 97. Parameters

Parameter	Description
Syntax1	If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqindex NULL,NULL,DBA` and `sp_iqindex Departments,NULL,DBA`.
Syntax2	You can specify the parameters in any order. Enclose them in single quotes.
Syntax 3 and 4	Produces slightly different output when a multicolumn index is present. Allows the same options as Syntax 1 and 2.

Description

Displays information about indexes in the database. Specifying one of the parameters returns the indexes from only that table, column, or tables owned by the specified user. Specifying more than one parameter filters the results by all of the parameters specified. Specifying no parameters returns all indexes for all tables in the database.

Table 98. sp_iqindex and sp_iqindex_alt columns

Column name	Description
table_name	The name of the table
table_owner	The owner of the table
column_name	The name of the column; multiple names can appear in a multicolumn index
index_type	The abbreviated index type (for example, **HG**, **LF**)
index_name	The name of the index
unique_index	'U' indicates the index is a unique index; otherwise, 'N'
location	TEMP = IQ temporary store, MAIN = IQ store, SYSTEM = catalog store
remarks	User comments added with the **COMMENT** statement

sp_iqindex always produces one line per index. **sp_iqindex_alt** produces one line per index per column if there is a multicolumn index.

See also

- *sp_iqcolumn Procedure* on page 447
- *sp_iqconstraint Procedure* on page 455
- *sp_iqdatatype Procedure* on page 463
- *sp_iqevent Procedure* on page 484
- *sp_iqhelp Procedure* on page 489
- *sp_iqpkeys Procedure* on page 536
- *sp_iqprocparm Procedure* on page 540
- *sp_iq_reset_identity Procedure* on page 549
- *sp_iqtable Procedure* on page 575
- *sp_iqview Procedure* on page 590

sp_iqindex and sp_iqindex_alt Procedure Examples

Use the example as a reference for **sp_iqindex** and **sp_iqindex_alt** usage.

The following variations in syntax both return all indexes on columns with the name `DepartmentID`:

```
call sp_iqindex (NULL,'DepartmentID')
```

```
sp_iqindex column_name='DepartmentID'
```

table_name	table_owner	column_name	index_type	index_name	unique_index	location	dbspace_id	remarks
Departments	GROUPO	DepartmentID	**FP**	ASIQ_IDX_T201_C1_FP	N	Main	16387	(NULL)
Departments	GROUPO	DepartmentID	**HG**	ASIQ_IDX_T201_C1_HG	U	Main	16387	(NULL)
Employees	GROUPO	DepartmentID	**FP**	ASIQ_IDX_T202_C5_FP	N	Main	16387	(NULL)

The following variations in syntax both return all indexes in the table `Departments` that is owned by table owner `GROUPO`:

```
sp_iqindex Departments,NULL,GROUPO
```

```
sp_iqindex table_name='Departments',table_owner='DBA'
```

table_name	table_owner	column_name	index_type	index_name	unique_index	location	dbspace_id	remarks
Departments	GROUPO	DepartmentHeadID	**FP**	ASIQ_IDX_T201_C3_FP	N	Main	16387	(NULL)
Departments	GROUPO	DepartmentID	**FP**	ASIQ_IDX_T201_C1_FP	N	Main	16387	(NULL)
Departments	GROUPO	DepartmentID	**HG**	ASIQ_IDX_T201_C1_HG	U	Main	16387	(NULL)
Departments	GROUPO	DepartmentName	**FP**	ASIQ_IDX_T201_C2_FP	N	Main	16387	(NULL)

The following variations in syntax for **sp_iqindex_alt** both return indexes on the table `Employees` that contain the column `City`. The index `emp_loc` is a multicolumn index on the columns `City` and `State`. **sp_iqindex_alt** displays one row per column for a multicolumn index.

```
sp_iqindex_alt Employees,City
```

```
sp_iqindex_alt table_name='Employees',
               column_name='City'
```

table_name	table_owner	column_name	index_type	index_name	unique_index	dbspace_id	remarks
Employees	GROUPO	City	**FP**	ASIQ_IDX_T452_C7_FP	N	16387	(NULL)
Employees	GROUPO	City	**HG**	emp_loc	N	16387	(NULL)
Employees	GROUPO	State	**HG**	emp_loc	N	16387	(NULL)

The output from **sp_iqindex** for the same table and column is slightly different:

```
sp_iqindex Employees,City
```

```
sp_iqindex table_name='Employee',column_name='City'
```

table_name	table_owner	column_name	index_type	index_name	unique_index	dbspace_id	location	remarks
Employees	GROUPO	City	**FP**	ASIQ_IDX_T452_C7_FP	N	16387	Main	(NULL)
Employees	GROUPO	City,State	**HG**	emp_loc	N	16387	Main	(NULL)

sp_iqindexadvice Procedure

Displays stored index advice messages. Optionally clears advice storage.

Syntax

sp_iqindexadvice `( [ resetflag ] )`

Privileges

Requires ALTER ANY INDEX or ALTER ANY OBJECT system privileges. Users without one of these system privileges must be granted `EXECUTE` permission to run the stored procedure.

Usage

Table 99. Parameters

Parameter	Description
resetflag	Lets the caller clear the index advice storage. If *resetflag* is nonzero, all advice is removed after the last row has been retrieved.

Description

Allows users to query aggregated index advisor messages using SQL. Information can be used to help decide which indexes or schema changes will affect the most queries.

INDEX_ADVISOR columns:

Table 100. sp_iqindexadvice columns

Column name	Description
Advice	Unique advice message
NInst	Number of instances of message
LastDT	Last date/time advice was generated

Examples

Sample output from the **sp_iqindexadvice** procedure:

Table 101. Sample sp_iqindexadvice output

Advice	NInst	LastDT
Add a CMP index on DBA.tb (c2, c3) Predicate: (tb.c2 = tb.c3)	2073	2009-04-07 16:37:31.000
Convert HG index on DBA.tb.c4 to a unique HG	812	2009-04-06 10:01:15.000
Join Key Columns DBA.ta.c1 and DBA.tb.c1 have mismatched data types	911	2009-02-25 20:59:01.000

See also

- *sp_iqcolumnuse Procedure* on page 450
- *sp_iqindexuse Procedure* on page 512
- *sp_iqtableuse Procedure* on page 580
- *sp_iqunusedcolumn Procedure* on page 585
- *sp_iqunusedindex Procedure* on page 586

- *sp_iqunusedtable Procedure* on page 587
- *sp_iqworkmon Procedure* on page 596

sp_iqindexfragmentation Procedure

Reports information about the percentage of page space taken up within the B-trees, garrays, and bitmap structures in SAP Sybase IQ indexes.

For garrays, the fill percentage calculation does not take into account the reserved space within the garray groups, which is controlled by the **GARRAY_FILL_FACTOR_PERCENT** option.

Syntax

```
dbo.sp_iqindexfragmentation ( 'target ' )
```

```
target: table table-name | index index-name [...]
```

Privileges

Requires the MANAGE ANY DBASPACE system privilege. Users without the MANAGE ANY DBSPACE system privilege must be granted `EXECUTE` permission to run the stored procedure.

Usage

Parameter	Description
table-name	Target **table** *table-name* reports on all nondefault indexes in the named table.
index-name	Target **index** *index-name* reports on the named index. Each index-name is a qualified index name. You can specify multiple indexes within the table, but you must repeat the **index** keyword with each index specified.

Example

Reports the internal index fragmentation for the unique **HG** index `DBA.prop_nu.prop_nu_a` table:.

Index	IndexType	Btree_Node_pa-ges	GAR-RAY_FILL_FAC-TOR_PERCENT
DBA.prop_nu.prop_nu_a	HG	8	25
SQLCODE:	0		

Index	IndexType	Btree_Node_pages	GARRAY_FILL_FACTOR_PERCENT
Fill Percent	btree pages	garray pages	bitmap pages
0-10%	13	2	8
11-20%	1	8	0
21-30%	0	4	0
31-40%	3	20	0
41-50%	4	116	0
51-60%	6	4	0
61-70%	3	3	0
71-80%	4	1	0
81-90%	1	1	0
91-100%	192	276	0

Note: All percentages are truncated to the nearest percentage point. **HG** indexes also display the value of option `GARRAY_FILL_FACTOR_PERCENT`. Index types that use a B-tree also display the number of node (nonleaf) pages. These are **HG**, **LF**, **WD**, **DATE**, and **DTTM**.

If an error occurrs during execution of this stored procedure, the SQLCODE would be nonzero.

See also

- *sp_iqindexmetadata Procedure* on page 505
- *sp_iqindexinfo Procedure* on page 503
- *sp_iqindexsize Procedure* on page 509
- *sp_iqrebuildindex Procedure* on page 544
- *sp_iqrowdensity Procedure* on page 553

sp_iqindexinfo Procedure

Displays the number of blocks used per index per main dbspace for a given object. If the object resides on several dbspaces, **sp_iqindexinfo** returns the space used in all dbspaces, as shown in the example.

Syntax

```
sp_iqindexinfo '{ database
| [ table table-name | index index-name ] [...] }
[ resources resource-percent ]'
```

Applies to

Simplex and multiplex.

Privileges

Requires MANAGE ANY DBSPACE system privilege. Users without MANAGE ANY DBSPACE system privilege must be granted `EXECUTE` permission to run the stored procedure.

Usage

You can request index information for the entire database, or you can specify any number of table or index parameters. If a table name is specified, **sp_iqindexinfo** returns information on all indexes in the table. If an index name is specified, only the information on that index is returned.

If the specified *table-name* or *index-name* is ambiguous or the object cannot be found, an error is returned.

By default in a multiplex database, **sp_iqindexinfo** displays information about the shared IQ store on a secondary node. If individual tables or indexes are specified, the store to display is automatically selected.

resource-percent must be an integer greater than 0. The resources percentage allows you to limit the CPU utilization of the **sp_iqindexinfo** procedure by specifying the percent of total CPUs to use.

Description

sp_iqindexinfo shows the DBA on which dbspaces a given object resides. The DBA can use this information to determine which dbspaces must be given **relocate** mode to relocate the object.

The results of **sp_iqindexinfo** are from the point of view of the version seen by the transaction running the command. Blocks used by other versions are not shown.

Table 102. sp_iqindexinfo columns

Column name	Description
Object	Table or index name
Dbspace_name	Name of the dbspace
ObjSize	Size of data for this object on this dbspace
DBSpPct	Percent of dbspace used by this object
MinBlk	First block used by this object on this dbspace
MaxBlk	Last block used by this object on this dbspace; useful for determining which objects must be relocated before the dbspace is resized to a smaller size

Examples

Displays information about indexes in the Departments table:

```
sp_iqindexinfo 'table GROUPO.Departments';
```

Object	Dbspace-Name	ObjSize	DBSpPct	MinBlk	MaxBlk
GROUPO.Departments	iq_main	288K	1	1,045,496.00	1,048,891.00
GROUPO.Departments.ASIQ_IDX_T779_C1_FP	iq_main	176K	1	1,047,197.00	1,047,328.00
GROUPO.Departments.ASIQ_IDX_T779_C2_FP	iq_main	160K	1	1,047,213.00	1,047,324.00
GROUPO.Departments.ASIQ_IDX_T779_C3_FP	iq_main	184K	1	1,047,229.00	1,047,317.00

Object	Dbspace-Name	ObjSize	DBSpPct	MinBlk	MaxBlk
GROUPO.Departments.ASIQ_IDX_T779_C3_HG	iq_main	440K	1	1,048,421.00	1,048,796.00
GROUPO.Departments.ASIQ_IDX_T779_I4_HG	iq_main	288K	1	1,047,261.00	1,047,306.00

See also

- *sp_iqdbspace Procedure* on page 467
- *sp_iqdbspaceinfo Procedure* on page 470
- *sp_iqspaceinfo Procedure* on page 560
- *sp_iqindexmetadata Procedure* on page 505
- *sp_iqindexfragmentation Procedure* on page 501
- *sp_iqindexsize Procedure* on page 509

sp_iqindexmetadata Procedure

Displays index metadata for a given index.

You can optionally restrict the output to only those indexes on a specified table, and to only those indexes belonging to a specified owner.

Syntax

```
dbo.sp_iqindexmetadata (index-name
[ , table-name [ , owner-name ] ] )
```

Privileges

User must be a table owner, have REFERENCE permissions on the table, or have either the ALTER ANY INDEX or ALTER ANY OBJECT system privilege.

Usage

Specifying a table name limits output to those indexes belonging to that table. Specifying an owner name limits output to indexes owned by that owner. Omitted parameters default to NULL. You can specify only one index per procedure.

User supplier IQ UNIQUE value for the column is available through sp_iqindexmetadata. It reports exact cardinality if Unique HG or LF are present. It reports 0 as cardinality if (only) non-unique HG is present.

Description

The first row of output is the owner name, table name, and index name for the index.

Subsequent rows of output are specific to the type of index specified.

Table 103. sp_iqindexmetadata output rows

Index type	Metadata returned
CMP, **DATE**, **DTTM**, **TIME**	Type, Version
FP	Type, Style, Version, DBType, Maximum Width, EstUnique, TokenCount, NBit, CountSize, DictSize, CountLen, MaxKeyToken, MinKey Token, MinCount, MaxCount, DistinctKey, IQ Unique
HG	Type, Version, Maintains Exact Distinct, Level 0 Threshold, Maximum Level Count,Tier ratio, Auto sizing, Average Load Size (records), Active Subindex count, Cardinality Range Min - Max, Estimated Cardinality, Accuracy of Cardinality
HNG	Type, Version, BitsPerBlockmap, NumberOfBits
LF	Type, Version, IndexStatus, NumberOfBlockmaps, BitsPerBlockmap, Distinct Keys
WD	Type, Version, KeySize, Delimiters, DelimiterCount, MaxKeyWordLength, PermitEmptyWord

Example

Value1	Value2	Value3
DBA	test2	test2_c1_hg
Type	HG	
Version	3	
Maintains Exact Distinct	No	
Level 0 Threshold	3000000	
Maximum Level Count	10	
Tier ratio	30	
Auto sizing	On	
Avarage Load Size (records)	58622	

Value1	Value2	Value3
Active Subindex count	3	
Cardinality Range Min - Max	5-5	
Estimated Cardinality	5	
Accuracy of Cardinality	100	
Level: 0 Main Index Total Row Count	1	
Level: 0 Main Index Deleted Row Count	0	
Level: 0 Main Index # of Btree Pages in Main btree	1	
Level: 0 Main Index # of Garray Pages	1	
Level: 0 Main Index # of Keys in Main Btree	1	
Level: 0 Main Index # of Keys Probed in Btree	0	
Level: 0 Main Index # of Keys Found Duplicate in Btree	0	
Level: 0 Main Index # of Keys Possible Distinct in Btree	0	
Level: 1 Main Index Total Row Count	3145747	
Level: 1 Main Index Deleted Row Count	0	
Level: 1 Main Index # of Btree Pages in Main btree	1	
Level: 1 Main Index # of Btree Pages in Conjugate btree	1	
Level: 1 Main Index # of Garray Pages	2	
Level: 1 Main Index # of Keys in Main Btree	8	
Level: 1 Main Index # of Keys in Conjugate Btree	3	

Value1	Value2	Value3
Level: 1 Main Index # rows in Conjugate Btree	2949127	
Level: 1 Main Index # of Keys Probed in Btree	0	
Level: 1 Main Index # of Keys Found Duplicate in Btree	0	
Level: 1 Main Index # of Keys Possible Distinct in Btree	0	
Level: 1 Incremental Index Total Row Count	1	
Level: 1 Incremental Index Deleted Row Count	0	
Level: 1 Incremental Index # of Btree Pages	1	
Level: 1 Incremental Index # of Garray Pages	1	
Level: 1 Incremental Index #of Keys in Btree	1	
Level: 1 Incremental Index # of Keys Probed in Btree	0	
Level: 1 Incremental Index # of Keys Found Duplicate in Btree	0	
Level: 1 Incremental Index # of Keys Possible Distinct in Btree	0	

See also

- *sp_iqindexfragmentation Procedure* on page 501
- *sp_iqindexinfo Procedure* on page 503
- *sp_iqindexsize Procedure* on page 509

sp_iqindexrebuildwidedata Procedure

Identifies wide columns in migrated databases that you must rebuild before they are available for read/write activities.

Syntax

sp_iqindexrebuildwidedata *[table.name]*

Privileges

User must be a table owner, have INSERT permission on a table to rebuild an index on that table, or have the INSERT ANY TABLE system privilege.

Description

CHAR, VARCHAR, BINARY, and VARBINARY columns wider than 255 characters, as well as all Long Varchar and Long Binary columns in databases migrated to SAP Sybase IQ 16.0 must be rebuilt before the database engine can perform read/write activities on them. **sp_iqindexrebuildwidedata** identifies these columns and generates a list of statements that you can use to rebuild the columns with the **sp_iqrebuildindex** procedure.

Include the optional *[table.name]* parameter to generate a list of wide columns for that table. Omit the *[table.name]* parameter to generate a list of wide columns for all tables in the database.

Example

Generate wide column rebuild statements for table T2:

```
sp_iqindexrebuildwidedata T2
```

Output:

```
Owner    Table    Column     Width    IndexType    sp_iqrebuild
DBA      T2       C1         256      1BitFP       sp_iqrebuildindex "DBA.T2"  column "C1" ;
```

sp_iqindexsize Procedure

Gives the size of the specified index.

Syntax

```
sp_iqindexsize [ [ owner.] table.] index_name
```

Privileges

User must be a table owner or have either the MANAGE ANY DBSPACE or ALTER ANY INDEX system privilege. Users without one of these system privileges must be granted `EXECUTE` permission to run the stored procedure.

Description

Table 104. sp_iqindexsize columns

Column name	Description
Username	Index owner.
Indexname	Index for which results are returned, including the table name.
Type	Index type.
Info	Component of the IQ index for which the KBytes, Pages, and Compressed Pages are being reported. The components vary by index type. For example, the default (FP) index includes BARRAY (barray) and Bitmap (bm) components. The Low_Fast (LF) index includes B-tree (bt) and Bitmap (bm) components.
KBytes	Physical object size in KB.
Pages	Number of IQ pages needed to hold the object in memory.
Compressed Pages	Number of IQ pages when the object is compressed (on disk).

Returns the total size of the index in bytes and kilobytes, and an Info column that describes the component of the IQ index for which the KBytes, Pages, and Compressed Pages are reported. The components described vary by index type. For example, the default (FP) index includes BARRAY (barray) and Bitmap (bm) components. The Low_Fast (LF) index includes B-tree (bt) and Bitmap (bm) components.

Also returns the number of pages required to hold the object in memory and the number of IQ pages when the index is compressed (on disk).

You must specify the *index_name* parameter with this procedure. To restrict results to this index name in a single table, include *owner.table.* when specifying the index.

Example

```
sp_iqindexsize ASIQ_IDX_T780_I4_HG
```

Username	Indexname	Type	Info	Kbytes	Pages	Compressed Pages
GROUPO	GROUPO.Departments.ASIQ_IDX_T780_I4_HG	HG	Total	288	4	2
GROUPO	GROUPO.Departments.ASIQ_IDX_T780_I4_HG	HG	vdo	0	0	0

User-name	Indexname	Type	Info	Kbytes	Pages	Com-pressed Pages
GROUPO	GROUPO.Departments.ASIQ_IDX_T780_I4_HG	HG	bt	152	2	1
GROUPO	GROUPO.Departments.ASIQ_IDX_T780_I4_HG	HG	garray	0	0	0
GROUPO	GROUPO.Departments.ASIQ_IDX_T780_I4_HG	HG	bm	136	2	1
GROUPO	GROUPO.Departments.ASIQ_IDX_T780_I4_HG	HG	barray	0	0	0
GROUPO	GROUPO.Departments.ASIQ_IDX_T780_I4_HG	HG	dpstore	0	0	0
GROUPO	GROUPO.Departments.ASIQ_IDX_T780_I4_HG	HG	large-lob	0	0	0
GROUPO	GROUPO.Departments.ASIQ_IDX_T780_I4_HG	HG	txtPst	0	0	0

```
CREATE TEXT INDEX ti ON Employees( Street ) IMMEDIATE
REFRESH;sp_iqindexsize 'ti';
```

User-name	Indexname	Type	Info	KBytes	Pages	Com-pressed Pages
GROUPO	GROUPO.Employees.ti	TEXT	Total	896	12	6
GROUPO	GROUPO.Employees.ti	TEXT	vdo	0	0	0
GROUPO	GROUPO.Employees.ti	TEXT	bt	304	4	2
GROUPO	GROUPO.Employees.ti	TEXT	garray	152	2	1
GROUPO	GROUPO.Employees.ti	TEXT	bm	136	2	1
GROUPO	GROUPO.Employees.ti	TEXT	barray	152	2	1

User-name	Indexname	Type	Info	KBytes	Pages	Com-pressed Pages
GROUPO	GROUPO.Employees.ti	TEXT	dpstore	0	0	0
GROUPO	GROUPO.Employees.ti	TEXT	large-lob	0	0	0
GROUPO	GROUPO.Employees.ti	TEXT	txtPst	304	4	2

See also

- *sp_iqindexmetadata Procedure* on page 505
- *sp_iqindexfragmentation Procedure* on page 501
- *sp_iqindexinfo Procedure* on page 503

sp_iqindexuse Procedure

Reports detailed usage information for secondary (non-FP) indexes accessed by the workload.

Syntax

```
sp_iqindexuse
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

Each secondary index accessed by the workload displays a row. Indexes that have not been accessed do not appear. Index usage is broken down by optimizer, constraint, and query usage.

Indexes from tables created in `SYSTEM` are not reported.

Table 105. sp_iqindexuse columns

Column name	Description
IndexName	Index name
TableName	Table name
Owner	User name of index owner
UID**	Index unique identifier

Column name	Description
Type	Index type
LastDT	Date/time of last access
NOpt	Number of metadata/uniqueness accesses
NQry	Number of query accesses
NConstraint	Number of accesses for unique or referential integrity checks

**UID is a number assigned by the system that uniquely identifies the instance of the index (where instance is defined when an object is created).

Example

Sample output from the **sp_iqindexuse** procedure.

```
IndexName       TableName  Owner UID Type LastDT                NOpt
 NQry  NConstraint
n_nationkey_hg  nation     DBA   29  HG   20070917
22:08:06~  12    0     12
n_regionkey_hg  nation     DBA   31  HG   20070917
22:08:06~  12    0      0
r_regionkey_hg  region     DBA   47  HG   20070917
22:08:06~  12    0     12
s_suppkey_hg    supplier   DBA   64  HG   20070917
22:08:06~  12    0     12
p_partkey_hg    part       DBA   87  HG   20070917
22:08:06~   6    0      6
s_suppkey_hg    supplier   DBA   64  HG   20070917
22:08:06~  12    0     12
...
```

See also

- *sp_iqcolumnuse Procedure* on page 450
- *sp_iqindexadvice Procedure* on page 499
- *sp_iqtableuse Procedure* on page 580
- *sp_iqunusedcolumn Procedure* on page 585
- *sp_iqunusedindex Procedure* on page 586
- *sp_iqunusedtable Procedure* on page 587
- *sp_iqworkmon Procedure* on page 596

sp_iqlmconfig Procedure

Controls license management configurations, displays and sets license type and status.

Syntax 1

```
sp_iqlmconfig 'edition', { 'SE' | 'SA' | 'EE' }
```

Table 106. Summary information for "edition" parameter

Topic	Value
Default	’EE’ (Enterprise Edition)
Range of values	’SE’ (Small Business) ’SA’ (Single Application) ’EE’ (Enterprise Edition)
Status	Static

Syntax 2

```
sp_iqlmconfig ’license type’, { ’CP’ | ’DT’ | ’SF’ | ’AC’ | ’BC’ |
'CH' | 'DH' | 'SH' | 'AH' | 'BH' }
```

Table 107. Summary information for "license type" parameter

Topic	Value
Default	’DT’ (Development and Testing)
Range of values	’AC’ (OEM CPU License) ’AH’ (OEM CPU License Chip) ’BC’ (OEM Standby License) ’BH’ (OEM Standby License Chip) ’CP’ (CPU License) ’CH’ (CPU License Chip) ’DH’ (Development and Testing License Chip) ’DT’ (Development and Testing) ’EV’ (Evaluation) ’SF’ (Standby CPU License) ’SH’ (Standby CPU License Chip)
Status	Static

Syntax 3

```
sp_iqlmconfig ’email severity’, { ’ERROR’ | ’WARNING’
| ’INFORMATIONAL’ | ’NONE’ }
```

NONE designates that e-mail notification is disabled.

Syntax 4

```
sp_iqlmconfig ’smtp host’, ’<host name>’ | ’
```

Table 108. Parameters

Parameter	Description
host name	Specifies SMTP host used for e-mail notification.

Syntax 5

```
sp_iqlmconfig ’email sender’, ’<email address>’ |
```

Table 109. Parameters

Parameter	Description
<email address>	Specifies the sender's e-mail address used for e-mail notification.

Syntax 6

```
sp_iqlmconfig 'email recipients', '<email recipients>' |
```

Table 110. Parameters

Parameter	Description
<email recipients>	Specifies a comma-separated list of e-mail addresses to whom e-mail notifications will be sent.

Syntax 7

```
sp_iqlmconfig |
```

Privileges

Requires the SERVER OPERATOR system privilege.

Usage

At startup, **sp_iqlmconfig** checks the edition type and license type of the license specified.

- If the specified license is not found, the server falls to grace mode.
- The specified license type becomes valid only when a non-null edition value is specified.
- If **sp_iqlmconfig** is called with no parameters (Syntax 3), it displays all the information above, as well as other information, such as:
  - Product Edition and License Type
  - Which of the optional licenses are in use
  - License count
  - E-mail information
  - General information about the license

See also

- *Properties Available for the Server* on page 101

sp_iqlocks Procedure

Shows information about locks in the database, for both the IQ main store and the IQ catalog store.

Syntax

```
sp_iqlocks ([connection,] [[owner.]table_name,] max_locks,]
[sort_order])
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Usage

Optional **sp_iqlocks** parameters you can specify to restrict results:

Parameter	Data Type	Description
connection	integer	Connection ID. With this option, the procedure returns information about locks for the specified connection only. Default is zero, which returns information about all connections.
owner.table_name	char(128)	Table name. With this option, the procedure returns information about locks for the specified table only. Default is NULL, which returns information about all tables in the database. If you do not specify owner, it is assumed that the caller of the procedure owns the table.
max_locks	integer	Maximum number of locks for which to return information. Default is 0, which returns all lock information.
sort_order	char(1)	Order in which to return information: • C sorts by connection (default) • T sorts by table_name

Description

Displays information about current locks in the database. Depending on the options you specify, you can restrict results to show locks for a single connection, a single table, or a specified number of locks.

sp_iqlocks displays the following information, sorted as specified in the sort_order parameter:

Column	Data type	Description
conn_name	VARCHAR(128)	The name of the current connection.
conn_id	INTEGER	Connection ID that has the lock.
user_id	CHAR(128)	User associated with this connection ID.
table_type	CHAR(6)	The type of table. This type is either BASE for a table, GLBTMP for global temporary table, or MVIEW for a materialized view. Materialized views are only supported for SQL Anywhere tables in the IQ catalog store.
creator	VARCHAR(128)	The owner of the table.
table_name	VARCHAR(128)	Table on which the lock is held.
index_id	INTEGER	The index ID or NULL
lock_class	CHAR(8)	String of characters indicating the type of lock: • S – share. • SW – share and write. • EW – exclusive and write. • E – exclusive. • P – phantom. • A – antiphantom. • W – write. All locks listed have one of S, E, EW, or SW, and may also have P, A, or both. Phantom and anti-phantom locks also have a qualifier of T or *: • T – the lock is with respect to a sequential scan. • * – the lock is with respect to all scans. • *nnn* – Index number; the lock is with respect to a particular index. SAP Sybase IQ obtains a share lock before a write lock. If a connection has exclusive lock, share lock does not appear. For write locks, if a connection has all-exclusive, share, and write locks, it is EW.
lock_duration	CHAR(11)	The duration of the lock. One of Transaction, Position, or Connection.

Column	Data type	Description
lock_type	CHAR(9)	Value identifying the lock (dependent on the lock class)
row_identifier	UNSIGNED BIGINT	The identifier for the row the lock starts on, or NULL.
row_range	BIGINT	The number of contiguous rows that are locked. Row locks in the RLV store can either be a single row, or a range of rows.

If **sp_iqlocks** cannot find the connection ID or user name of the user who has a lock on a table, it displays a 0 (zero) for the connection ID and `User unavailable` for the user name.

Note: Exclusive, phantom, or antiphantom locks can be placed on IQ catalog store tables, but not on SAP Sybase IQ tables in the IQ main store. Unless you have explicitly taken out locks on a table in the catalog store, you never see these types of locks (or their qualifiers T, *, and *nnn*) in a SAP Sybase IQ database.

Examples

The example shows the **sp_iqlocks** procedure call and its output in the SAP Sybase IQ database. The procedure is called with all default options, so that the output shows all locks, sorted by connection.

```
call sp_iqlocks()
```

```
conn_name         conn_id
user_id  table_type  creator  table_name
=========         =======
=======  ==========  =======  ==========
SQL_DBC_13cd6038  3          DBA     BASE       DBA     rv_locks2
SQL_DBC_13cd6038  3          DBA     BASE       DBA     rv_locks2
SQL_DBC_13cd6038  3          DBA     BASE       DBA     rv_locks2
RVL_CONN_T775    1000000407          BASE       DBA     rv_locks2

index_id  lock_class  lock_duration  lock_type  row_identifier
row_range
========  ==========  =============  =========  ==============
=========
          Schema      Transaction    Shared
          Row         Transaction    Row        1                4
          Row         Transaction    Row        281474976710656  1
          Table       Transaction    Intent
```

sp_iqmergerlvstore Procedure

Triggers a merge of the row-level versioned (RLV) store with the IQ main store.

Syntax

```
sp_iqmergerlvstore (merge_type, table_name, [ table_owner ])
```

Usage

- If a table name is not specified, all the active data (from all RLV-enabled tables) in the RLV store will be merged into the IQ main store.
- Merge-type can be **BLOCKING | NON-BLOCKING** . (Of the rare cases where you may wish to run a manual merge, almost all would be blocking merges.)
- After performing the merge, the stored procedure will automatically commit the merge transaction.

sp_iqmodifylogin Procedure

Assigns a user to a login policy.

Syntax1

```
call sp_iqmodifylogin 'userid', ['login_policy_name']
```

Syntax2

```
sp_iqmodifylogin 'userid', ['login_policy_name']
```

Privileges

Requires the MANAGE ANY USER system privilege.

Usage

Table 111. Parameters

Parameter	Description
userid	Variable that holds the name of the account to modify.
login_policy_name	(Optional) Specifies the name of the login policy to which the user will be assigned. If no login policy name is specified, the user is assigned to the root login policy.

Examples

Assigns user `joe` to a login policy named `expired_password`:

```
sp_iqmodifylogin 'joe', 'expired_password'
```

Assigns user `joe` to the root login policy:

```
call sp_iqmodifylogin ('joe')
```

See also

- *sp_expireallpasswords system procedure* on page 668
- *sp_iqaddlogin Procedure* on page 429
- *sp_iqcopyloginpolicy Procedure* on page 459
- *sp_iqpassword Procedure* on page 534

sp_iqmpxcheckdqpconfig Procedure

sp_iqmpxcheckdqpconfig is a diagnostic tool that checks the DQP configuration for the current connection. If DQP fails, run **sp_iqmpxcheckdqpconfig** to determine if DQP configuration issues are causing the query distribution failure.

Syntax

```
sp_iqmpxcheckdqpconfig
```

Applies to

Multiplex only.

Privileges

No special privileges are required to execute the procedure.

Description

Table 112. Column Descriptions

Column Name	Description
DiagMsgID	Uniquely identifies a diagnostic message
Description	Diagnostic message describing the issue found with DQP configuration

Table 113. Diagnostic Messages

DiagMsgID	Description
0	`No issues found with DQP configuration`
1	`Database is a simplex`

DiagMsgID	Description
2	Multiplex is running in single-node configuration mode
3	Logical server policy option dqp_enabled is set to 0
4	Temporary dqp_enabled connection option is set to OFF
5	Logical server context has only one member node
6	Coordinator does not participate in DQP since its named membership in the logical server is currently ineffective
7	Coordinator does not participate in DQP since its logical membership in the logical server is currently ineffective because ALLOW_COORDINATOR_AS_MEMBER option in Root Logical server policy set to OFF
8	There is no dbfile in IQ_SHARED_TEMP dbspace
9	All dbfiles in IQ_SHARED_TEMP dbspace are READ ONLY
10	IQ_SHARED_TEMP dbspace is dynamically offline

Example

Sample output from the **sp_iqmpxcheckdqpconfig** procedure:

```
diagmsgid          description
3               Logical server policy option dqp_enabled is set to 0
5                  Logical server context has only one member node
6                  Coordinator does not participate in DQP since its
                   named membership in the logical server is
                   currently ineffective
```

```
7                  Coordinator does not participate in DQP since
                   its logical membership in the logical server
                   is currently ineffective because
                   ALLOW_COORDINATOR_AS_MEMBER option in Root
                   Logical server policy set to OFF
8                  There is no dbfile in IQ_SHARED_TEMP dbspace
```

sp_iqmpxdumptlvlog Procedure

Returns the contents of the table version log in a readable format.

Syntax

sp_iqmpxdumptlvlog
[main], [asc | desc]

Applies to

Multiplex only.

Privileges

Requires MANAGE MULTIPLEX system privilege. Users without the MANAGE MULTIPLEX system privilege must be granted EXECUTE permission.

Description

sp_iqmpxdumptlvlog returns the contents of the queue through which the coordinator propagates DML and DDL commands to secondary nodes.

The **asc** or **desc** arguments specify the row order. These arguments require the **main** argument. The default options are:

```
'main', 'asc'.
```

Examples

Show the output of **sp_iqmpxdumptlvlog**:

```
RowID     Contents
--------------------------------------------------------------
1         Txn CatId:196 CmtId:196 TxnId:195 Last Rec:1
          UpdateTime: 2011-08-08 15:41:43.621
2         Txn CatId:243 CmtId:243 TxnId:242 Last Rec:5
          UpdateTime: 2011-08-08 15:42:25.070
3         DDL: Type=34, CatID=0, IdxID=0,
          Object=IQ_SYSTEM_TEMP, Owner=mpx4022_w1
4         CONN: CatID=0, ConnUser=
5         SQL: ALTER DBSPACE "IQ_SYSTEM_TEMP" ADD FILE
          "w1_temp1" '/dev/raw/raw25' FILE ID 16391 PREFIX 65536
          FINISH 0 FIRST BLOCK
1         BLOCK COUNT 3276792 RESERVE 0 MULTIPLEX SERVER
          "mpx4022_w1" COMMITID 242 CREATETIME
          '2011-08-08 15:42:24.860'
6         Txn CatId:283 CmtId:283 TxnId:282 Last Rec:7
          UpdateTime: 2011-08-08 15:42:50.827
7         RFRB TxnID: 242 CmtID:243 ServerID 0 BlkmapID:
```

```
          0d00000000000000d2000a000000000020000000000000000
          0000000000000000000008003501010000000c38000000000000
          010000000000000000000000RFID:01000501000000001300000
          0000000001000000000000100RBID:0100050100000000013000
```

sp_iqmpxfilestatus Procedure

If run on the coordinator node, displays file status for coordinator and for every shared dbspace file on every included secondary node. If executed on a secondary node, displays file status for only the current node.

Syntax

```
sp_iqmpxfilestatus
```

Applies to

Multiplex only.

Privileges

Must have the MANAGE MULTIPLEX system privilege. Users without the MANAGE MULTIPLEX system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

sp_iqmpxfilestatus returns:

Column Name	Data Type	Description
ServerID	unsigned int	Identifier for the multiplex server, from SYSIQMPXINFO
DBSpaceName	char(128)	Dbspace from which the space is reserved
FileName	char(128)	Logical file name of the dbspace file
FileStatus	char(2)	Dbspace file status: • VALID – file path and permissions are correct • INVALID_PATH – path name not accessible • INVALID_PERM – file permissions are incorrect

Example

Shows sample output of **sp_iqmpxfilestatus**:

```
server_id,server_name,DBSpace_name,FileName,FileStatus
1,'mpx2422_m','IQ_SYSTEM_MAIN','IQ_SYSTEM_MAIN','VALID'
1,'mpx2422_m','mpx_main1','mpx_main1','VALID'
1,'mpx2422_m','IQ_SHARED_TEMP','sharedfile_dba','VALID'
1,'mpx2422_m','IQ_SHARED_TEMP','sharedfile_dba1','VALID'
2,'mpx2422_w1','IQ_SYSTEM_MAIN','IQ_SYSTEM_MAIN','VALID'
2,'mpx2422_w1','mpx_main1','mpx_main1','VALID'
2,'mpx2422_w1','IQ_SHARED_TEMP','sharedfile_dba','VALID'
2,'mpx2422_w1','IQ_SHARED_TEMP','sharedfile_dba1','VALID'
3,'mpx2422_r1','IQ_SYSTEM_MAIN','IQ_SYSTEM_MAIN','VALID'
3,'mpx2422_r1','mpx_main1','mpx_main1','VALID'
3,'mpx2422_r1','IQ_SHARED_TEMP','sharedfile_dba','VALID'
3,'mpx2422_r1','IQ_SHARED_TEMP','sharedfile_dba1','VALID'
```

sp_iqmpxincconnpoolinfo Procedure

If run on the coordinator node, displays INC connection pool status for every node. If executed on a secondary node, displays INC connection pool status for only the current node.

Syntax

```
sp_iqmpxincconnpoolinfo
```

Applies to

Multiplex only.

Usage

If the procedure is run on the coordinator and a secondary node is not responding or has timed out, the result set omits the row for that node, because this data cannot be accessed unless that node is running.

Privileges

MANAGE MULTIPLEX system privilege required. Users without the MANAGE MULTIPLEX system privilege must be granted `EXECUTE` permission.

Description

The **sp_iqmpxincconnpoolinfo** procedure returns:

Column Name	Data Type	Description
server_id	unsigned int	Identifier for the server
server_name	char(128)	Name of the server
current_pool_size	unsigned int	Current size of connection pool
idle_connection_count	unsigned int	Number of idle connections in the pool
connections_in_use	unsigned int	Number of connections in use

Example

Shows sample output of **sp_iqmpxincconnpoolinfo**:

```
server_id,server_name,current_pool_size,
idle_connection_count,connections_in_use

2,'r2_dbsrv90210',0,0,0

3,'w3_dbsrv90210',0,0,0
```

sp_iqmpxincheartbeatinfo Procedure

If run on the coordinator node, displays INC heartbeat status for every node. If executed on a secondary node, displays INC heartbeat status for just the current node.

Syntax

```
sp_iqmpxincheartbeatinfo
```

Applies to

Multiplex only.

Privileges

MANAGE MULTIPLEX system privilege required. Users without the MANAGE MULTIPLEX system privilege must be granted `EXECUTE` permission.

Description

The **sp_iqmpxincheartbeatinfo** procedure returns:

Column Name	Data Type	Description	Values
server_id	unsigned int	Identifier for the server	
server_name	char(128)	Name of the server	
last_positive_hb	TIMESTAMP	Date/time of last successful heartbeat ping	DD:MM:YYYY:HH:MM:SS
time_not_responding	TIME	Time since last successful heartbeat ping	HH:MM:SS
time_until_timeout	TIME	If a node is not responding, the time left until node is declared offline.	

sp_iqmpxincheartbeatinfo Examples

Sample output of **sp_iqmpxincheartbeatinfo**.

```
server_id,server_name,last_positive_hb,
time_not_responding,time_until_timeout
```

```
2,'r2_dbsrv90210',2012-11-17
15:48:42.0,00:00:00,00:00:00
3,'w3_dbsrv90210',2012-11-17
15:48:42.0,00:00:00,00:00:00
```

If the elapsed time exceeds 24 hours, SAP Sybase IQ returns **sp_iqmpxincheartbeatinfo** output like the following:

```
server_id,server_name,last_positive_hb,
time_not_responding,time_until_timeout
2,'r2_mpx_cr_srv',Jan 14 2013 11:57AM,11:59PM,11:59PM
3,'w4_mpx_cr_srv',Jan 14 2013
11:57AM,11:59PM,11:59PM
(2 rows affected)
(return status = 0)
```

A value of 11:59PM in the `time_not_responding` and `time_until_timeout` columns means that the time has crossed the 24-hour limit.

sp_iqmpxincstatistics Procedure

Displays a snapshot of the aggregate statistics of internode communication (INC) status since server startup as of the moment of execution.

Syntax

sp_iqmpxincstatistics

Applies to

Multiplex only.

Privileges

Requires the MANAGE ANY STATISTICS system privilege. Users without MANAGE ANY STATISTICS system privilege must be granted `EXECUTE` permission.

Description

Returns:

Table 114. sp_iqmpxincstatistics Columns

Column Name	Data Type	Description
stat_name	char(128)	INC statistics name
stat_value	unsigned integer	INC statistics value

Table 115. Valid stat_name Values

Value	Description
NumSuspendedINC	Number of suspended INC connections since server startup
NumResumedINC	Number of resumed INC connections since server startup
NumDroppedSuspendedINC	Number of dropped INC connections that have been suspended (on coordinator only)
NumSuspendedTxnRollbackINC	Number of rolled back global DML transactions due to INC failure (on writer only)

Example

Shows one suspended and one resumed transaction:

```
sp_iqmpxincstatistics
stat_name                     stat_value
NumSuspendedINC               1
NumResumedINC                 1
NumSuspendedTXNRollBackINC    0
```

sp_iqmpxinfo Procedure

Returns a row for every node in the multiplex. Can be run from any multiplex node.

Syntax

```
sp_iqmpxinfo
```

Applies to

Multiplex only.

Privileges

MANAGE MULTIPLEX system privilege required.

Description

The **sp_iqmpxinfo** procedure returns:

Column Name	Data Type	Description
server_id	unsigned int	Identifier for the server for which information appears
server_name	char(128)	Name of the server

Column Name	Data Type	Description
connection_info	long varchar	A formatted string containing the host/port portion of the connection string used for TCP/IP connections between multiplex servers.
db_path	long varchar	Full database path
role	char(16)	'coordinator' | 'writer' | 'reader'
status	char(8)	'included' | 'excluded'
mpx_mode	char(16)	'single' | 'coordinator' | 'writer' | 'reader' | 'unknown'
inc_state	char(16)	'active' | 'not responding' | 'timed out'
coordinator_failover	char(128)	Name of the failover server
current_version	unsigned bigint	Decimal-formatted version ID
active_versions	long_varchar	Comma-separated list of decimal formatted version IDs.
private_connection_in-fo	long varchar	A formatted string containing the host/port portion of the connection string used for private TCP/IP connections between multiplex servers
mipc_priv_state	char(16)	'active' – MIPC connection to this node is active over the private interconnect | 'not responding' – MIPC connection to this node is not responding over pri-vate interconnect.
mipc_public_state	char(16)	'active' – MIPC connection to this node is active over the public interconnect. | 'not responding' – MIPC connection to this node is not responding over public interconnect.

Example

Sample output of **sp_iqmpxinfo** :

```
server_id,server_name,connection_info,db_path,role,
status,mpx_mode,inc_state,coordinator_failover,
current_version,active_versions,private_connection_
info,mipc_priv_state,mipc_public_state

1,'my_mpx1','host=(fe80::214:4fff:fe45:be26%2):1362
0,(fd77:55d:59d9:329:214:4fff:fe45:be2
6%2):13620,10.18.41.196:13620','/system3/users
/devices/s16900269/iqmpx1/mpx1.db',
'coordinator','included','coordinator','N/A',
'my_mpx2',0,,,'active','active'
```

```
2,'IQ_mpx2','host=system3:13625',
'/system3/users/devices/s16900269
/iqmpx_2/wk0001.db','writer','included',
'writer','active','IQ_mpx20', 'not responding','active'

3,'IQ_mpx3,'host=system3:13630/system3/users/devi
ces/s16900269/iqmpx_3/mpx1.db','reader','included',
'unknown',timed out',
'IQ_mpx20','not responding',
'not responding'
```

sp_iqmpxsuspendedconninfo Procedure

Shows details about currently suspended connections and transactions on the coordinator node.

Syntax

sp_iqmpxsuspendedconninfo

Applies to

Multiplex only.

Privileges

Requires any of the following system privileges to see all suspended connections in the database:

- DROP CONNECTION system privilege
- MONITOR system privilege
- SERVER OPERATOR system privilege

No system privileges are required to see a user's own suspended connections. This procedure is owned by user DBO.

Description

Returns:

Table 116. sp_iqmpxsuspendedconninfo Columns

Column Name	Data Type	Description
ConnName	char(128)	Connection name
ConnHandle	unsigned integer	Connection identifier
GlobalTxnID	unsigned bigint	Global transaction identifier of active transaction on this connection

Column Name	Data Type	Description
MPXServerName	char(128)	Name of the multiplex server where the INC connection originates
TimeInSuspendedState	integer	Total time, in seconds, spent by the connection in suspended state
SuspendTimeout	integer	Suspend timeout, in seconds (2*MPX_LIVENESS_TIME-OUT)

Example

sp_iqmpxsuspendedconninfo

```
ConnName              ConnHandle      GlobalTxnId
===================   =============   =============
'IQ_MPX_SERVER_P54'              14          112753

MPXServerName         TimeInSuspendedState
==================    =======================
'HP1_12356_IQ_mpx2'                        37

SuspendTimeout
===============
            360
```

sp_iqmpxvalidate Procedure

Checks multiplex configuration for inconsistencies.

Syntax

call dbo.sp_iqmpxvalidate `( 'show_msgs' )`

Applies to

Multiplex only.

Privileges

None.

Description

Executes multiple checks on tables `SYS.SYSIQDBFILE` and other multiplex events and stored procedures. May run on any server. Returns a severity result to the caller; values are:

Value	Description
0	No errors detected
1	Dynamic state is not as expected.

Value	Description
2	Nonfatal configuration error; for example, multiplex operation impaired
3	Fatal configuration problem; for example, one or more servers might not start

If called interactively, also returns a table of the errors found, if any, unless the calling parameter is not 'Y'.

Each error indicates its severity. If there are no errors, the procedure returns `No errors detected`.

sp_iqmpxversioninfo Procedure

Shows the current version information for this server, including server type (write server, query server, single-node mode) and synchronization status.

Syntax

```
sp_iqmpxversioninfo
```

Applies to

Multiplex only.

Privileges

No specific system privilege is required.

Description

Table 117. sp_iqmpxversioninfo Columns

Column	Data Type	Description
CatalogID	unsigned bigint	Catalog version on this server
VersionID	unsigned bigint	Latest version available on this server
OAVID	unsigned bigint	Oldest active version on this server
ServerType	char(1)	Type of server: “C” (Coordinator), “W” (Write Server) or “Q” (Query Server)
CatalogSync	char(1)	Catalog synchronization: “T” (synchronized) or “F” (not synchronized)
WCatalogID	unsigned bigint	Catalog version on the write server
WVersionID	unsigned bigint	Latest version available on the write server

sp_iqobjectinfo Procedure

Returns partitions and dbspace assignments of database objects and subobjects.

Syntax

sp_iqobjectinfo [*owner_name*] [, *object_name*] [, *object-type*]

Privileges

No specific system privilege is required to run this procedure.

Usage

Table 118. Parameter

Parameter	Description
owner_name	Owner of the object. If specified, **sp_iqobjectinfo** displays output only for tables with the specified owner. If not specified, **sp_iqobjectinfo** displays information on tables for all users in the database.
object_name	Name of the table. If not specified, **sp_iqobjectinfo** displays information on all tables in the database.
object-type	Valid **table** object types. If the object-type is a table, it must be enclosed in quotation marks.

All parameters are optional, and any parameter may be supplied independent of the value of another parameter.

Use input parameters with **sp_iqobjectinfo**; you can query the results of the **sp_iqobjectinfo** and it performs better if you use input parameters rather than using predicates in the **WHERE** clause of the query. For example, Query A is written as:

```
SELECT COUNT(*) FROM sp_iqobjectinfo()
WHERE owner = 'DBA'
AND object_name = 'tab_case510'
AND object_type = 'table'
AND sub_object_name is NULL
AND dbspace_name = 'iqmain7'
AND partition_name = 'P1'
```

Query B is Query A rewritten to use **sp_iqobjectinfo** input parameters:

```
SELECT COUNT(*) FROM sp_iqobjectinfo('DBA','tab_case510','table')
WHERE sub_object_name is NULL
AND dbspace_name = 'iqmain7'
AND PARTITION_NAME = 'P1'
```

Query B returns results faster than Query A. When the input parameters are passed to **sp_iqobjectinfo**, the procedure compares and joins fewer records in the system tables, thus doing less work compared to Query A. In Query B, the predicates are applied in the procedure itself, which returns a smaller result set, so a smaller number of predicates is applied in the query.

The **sp_iqobjectinfo** stored procedure supports wildcard characters for interpreting *owner_name*, *object_name*, and *object_type*. It shows information for all dbspaces that match the given pattern in the same way the **LIKE** clause matches patterns inside queries.

Description

Returns all the partitions and the dbspace assignments of a particular or all database objects (of type table) and its subobjects. The subobjects are columns, indexes, primary key, unique constraints, and foreign keys.

Table 119. sp_iqobjectinfo columns

Column Name	Description
owner	Name of the owner of the object.
object_name	Name of the object (of type table) located on the dbspace.
sub_object_name	Name of the object located on the dbspace.
object_type	Type of the object (column, index, primary key, unique constraint, foreign key, partition, or table).
object_id	Global object ID of the object.
id	Table ID of the object.
dbspace_name	Name of the dbspace on which the object resides. The string "[multiple]" appears in a special meta row for partitioned objects. The [multiple] row indicates that multiple rows follow in the output to describe the table or column.
partition_name	Name of the partition for the given object.

Examples

Note: These examples show objects in the `iqdemo` database to better illustrate output. `iqdemo` includes a sample user dbspace named `iq_main` that may not be present in your own databases.

Displays information about partitions and dbspace assignments of a specific database object and subobjects owned by a specific user:

```
sp_iqobjectinfo GROUPO,Departments
```

```
owner   object_name  sub_object_name                object_type  obj
ect_id  id
GROUPO  Departments  (NULL)                          table         3
```

```
632     738
GROUPO  Departments  DepartmentID                   column        3
633     738
GROUPO  Departments  DepartmentName                 column        3
634     738
GROUPO  Departments  DepartmentHeadID               column        3
635     738
GROUPO  Departments  DepartmentsKey                 primary
key   83       738
GROUPO  Departments  FK_DepartmentHeadID_EmployeeID  foreign
key   92       738

dbspace_name     partition_name
iq_main           (NULL)
iq_main           (NULL)
iq_main           (NULL)
iq_main           (NULL)
iq_main           (NULL)
iq_main           (NULL)
```

Displays information about partitions and dbspace assignments of a specific database object and subobjects owned by a specific user for *object-type* **table**:

```
sp_iqobjectinfo DBA,sale,'table'
```

```
owner  object_name  sub_object_name  object_type  object_id  id
DBA    sale          (NULL)           table          3698     742
DBA    sale         prod_id           column         3699     742
DBA    sale         month_num         column         3700     742
DBA    sale         rep_id            column         3701     742
DBA    sale         sales             column         3702     742

dbspace_name      partition_name
iq_main            (NULL)
iq_main            (NULL)
iq_main            (NULL)
iq_main            (NULL)
iq_main            (NULL)
```

sp_iqpassword Procedure

Changes a user's password.

Syntax1

call sp_iqpassword (`'caller_password'`, `'new_password'` [, `'user_name'`])

Syntax2

sp_iqpassword `'caller_password'`, `'new_password'` [, `'user_ name'`]

Privileges

- None to set your own password.
- The CHANGE PASSWORD system privilege is required to set other users' passwords.

Usage

Table 120. Parameters

Parameter	Description
caller_password	Your password. When you are changing your own password, this is your old password. When a user with the CHANGE PASSWORD system priv-ielge is changing another user's password, call-er_password is the of the user making the change.
new_password	New password for the user, or for *loginname*.
user_name	Login name of the user whose password is being changed by by another user with CHANGE PASSWORD system privilege. Do not specify user_name when changing your own password.

Description

A user password is an identifier. Any user can change his or her own password using **sp_iqpassword**. The CHANGE PASSWORD system privilege is required to change the password of any existing user.

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

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

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

Examples

Changes the password of the logged-in user from irk103 to exP984:

```
sp_iqpassword 'irk103', 'exP984'
```

If the logged-in user has the CHANGE PASSWORD system privilege or `joe`, the password of user `joe` from `eprr45` to `pdi032`:

```
call sp_iqpassword ('eprr45', 'pdi932', 'joe')
```

See also

- *sp_expireallpasswords system procedure* on page 668

- *sp_iqaddlogin Procedure* on page 429
- *sp_iqcopyloginpolicy Procedure* on page 459
- *sp_iqmodifylogin Procedure* on page 519

sp_iqpkeys Procedure

Displays information about primary keys and primary key constraints by table, column, table owner, or for all SAP Sybase IQ tables in the database.

Syntax

```
sp_iqpkeys { [ table-name ], [ column-name ], [ table-owner ] }
```

Privileges

No specific system privilege is required to run this procedure.

Usage

Parameter	Description
table-name	The name of a base or global temporary table. If specified, the procedure returns information about primary keys defined on the specified table only.
column-name	The name of a column. If specified, the procedure returns information about primary keys on the specified column only.
table-owner	The owner of a table or table. If specified, the procedure returns information about primary keys on tables owned by the specified owner only.

One or more of the parameters can be specified. If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. If none of the parameters are specified, a description of all primary keys on all tables in the database is displayed. If any of the specified parameters is invalid, no rows are displayed in the output.

Table 121. sp_iqpkeys usage examples

Syntax	Output
sp_iqpkeys sales	Displays information about primary keys defined on table `sales`
sp_iqpkeys sales, NULL, DBA	Displays information about primary keys defined on table `sales` owned by `DBA`

Syntax	Output
sp_iqpkeys sales, store_id, DBA	Displays information about primary key defined on column `store_id` of table `sales` owned by `DBA`
sp_iqpkeys NULL, NULL, DBA	Displays information about primary keys defined on all tables owned by `DBA`

Description

The **sp_iqpkeys** stored procedure displays the following information about primary keys on base and global temporary tables in a database:

Table 122. sp_iqpkeys columns

Column name	Description
table_name	The name of the table
table_owner	The owner of the table
column_name	The name of the column(s) on which the primary key is defined
column_id	The column ID
constraint_name	The name of the primary key constraint
constraint_id	The primary key constraint ID

Examples

Display the primary keys defined on columns of table `sales1`:

```
sp_iqpkeys sales1

table_name table_owner column_name column_id constraint_name constra
int_id
sales1         DBA        store_id     1         MA114           114
```

Display the primary keys defined on columns of table `sales2`:

```
sp_iqpkeys sales2

table_name table_owner column_name column_id constraint_name constra
int_id
sales2         DBA      store_id,     1,2        MA115           115
                        order_num
```

Display the primary keys defined on the column `store_id` of table `sales2`:

```
sp_iqpkeys sales2, store_id

table_name table_owner column_name column_id constraint_name constra
int_id
sales2         DBA      store_id       1         MA115           115
```

See also

- *sp_iqcolumn Procedure* on page 447
- *sp_iqconstraint Procedure* on page 455
- *sp_iqdatatype Procedure* on page 463
- *sp_iqevent Procedure* on page 484
- *sp_iqhelp Procedure* on page 489
- *sp_iqindex and sp_iqindex_alt Procedures* on page 496
- *sp_iqprocparm Procedure* on page 540
- *sp_iq_reset_identity Procedure* on page 549
- *sp_iqtable Procedure* on page 575
- *sp_iqview Procedure* on page 590

sp_iqprocedure Procedure

Displays information about system and user-defined procedures.

Syntax

```
sp_iqprocedure [ proc-name ], [ proc-owner ], [ proc-type ]
```

Privileges

No specific system privilege is required to run this procedure.

Usage

Table 123. Parameters

Parameter	Description
proc-name	The name of the procedure.
proc-owner	The owner of the procedure.
proc-type	The type of procedure. Allowed values are: • **SYSTEM**: displays information about system procedures (procedures owned by user `SYS` or `dbo`) only • **ALL**: displays information about user and system procedures • Any other value: displays information about user procedures

The **sp_iqprocedure** procedure can be invoked without any parameters. If no parameters are specified, only information about user-defined procedures (procedures not owned by `dbo` or `SYS`) is displayed by default.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqprocedure NULL, NULL, SYSTEM` and `sp_iqprocedure NULL, user1`.

Table 124. sp_iqprocedure usage examples

Syntax	Output
sp_iqprocedure	Displays information about all procedures in the database not owned by `dbo` or `SYS`
sp_iqprocedure sp_test	Displays information about the procedure **sp_test**
sp_iqprocedure non_existing_proc	No rows returned, as the procedure **non_existing_proc** does not exist
sp_iqprocedure NULL, DBA	Displays information about all procedures owned by `DBA`
sp_iqprocedure sp_test, DBA	Displays information about the procedure **sp_test** owned by `DBA`
sp_iqprocedure sp_iqtable	The procedure **sp_iqtable** is not a system procedure. If there is no user-defined procedure also named **sp_iqtable**, no rows are returned. (By default only user-defined procedures are returned.)
sp_iqprocedure sp_iqtable, dbo	No rows returned, as the procedure **sp_iqtable** is not a user procedure (by default only user procedures returned)
sp_iqprocedure NULL, NULL, SYSTEM	Displays information about all system procedures (owned by `dbo` or `SYS`)
sp_iqprocedure sp_iqtable, NULL, SYSTEM	Displays information about the system procedure **sp_iqtable**
sp_iqprocedure sp_iqtable, dbo, ALL	Displays information about the system procedure **sp_iqtable** owned by `dbo`

Description

The **sp_iqprocedure** stored procedure displays information about procedures in a database. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if *proc-name* is specified, only information about the specified procedure is displayed. If *proc-owner* is specified, **sp_iqprocedure** returns only information about procedures owned by the specified owner. If no parameters are specified, **sp_iqprocedure** displays information about all the user-defined procedures in the database.

The **sp_iqprocedure** procedure returns information in the following columns:

Table 125. sp_iqprocedure columns

Column name	Description
proc_name	The name of the procedure
proc_owner	The owner of the procedure
proc_defn	The command used to create the procedure. For hidden procedures, the keyword 'HIDDEN' is displayed.
replicate	Displays Y if the procedure is a primary data source in a Replication Server installation; N if not.
srvid	Indicates the remote server, if the procedure is on a remote database server
remarks	A comment string

Examples

Displays information about the user-defined procedure `sp_test`:

```
sp_iqprocedure sp_test

proc_name    proc_owner    proc_defn       replicate     srvid     r
emarks

sp_test     DBA       create procedure   N           (NULL)    (NULL)
                        DBA.sp_test(in n1
                        integer)
                        begin message'sp_test'end
```

Displays information about all procedures owned by user `DBA`:

```
sp_iqprocedure NULL, DBA

proc_name    proc_owner    proc_defn       replicate     srvid     r
emarks

sp_test     DBA       create procedure   N           (NULL)    (NULL)
                        DBA.sp_test(in n1
                        integer)
                        begin message'sp_test'end
sp_dept     DBA       create procedure   N           (NULL)    (NULL)
                        DBA.sp_dept() begin end
```

sp_iqprocparm Procedure

Displays information about stored procedure parameters, including result set variables and SQLSTATE/SQLCODE error values.

Syntax

sp_iqprocparm [*proc-name*], [*proc-owner*], [*proc-type*]

Privileges

No specific system privilege is required to run this procedure.

Usage

Table 126. Parameters

Parameter	Description
proc-name	The name of the procedure.
proc-owner	The owner of the procedure.
proc-type	The type of procedure. Allowed values are: • **SYSTEM**: displays information about system procedures (procedures owned by user `SYS` or `dbo`) only • **ALL**: displays information about user and system procedures • Any other value: displays information about user procedures

You can invoke **sp_iqprocparm** without parameters. If you do not specify any parameters, input/output and result parameters of user-defined procedures (procedures not owned by `dbo` or `SYS`) appear.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqprocparm NULL, NULL, SYSTEM` and `sp_iqprocparm NULL, user1`.

Table 127. sp_iqprocparm usage examples

Syntax	Output
sp_iqprocparm	Displays parameters for all procedures in the database not owned by `dbo` or `SYS`
sp_iqprocparm sp_test	Displays information about the procedure **sp_test**
sp_iqprocparm non_existing_proc	No rows returned, as the procedure **non_existing_proc** does not exist
sp_iqprocparm NULL, DBA	Displays parameters for all procedures owned by `DBA`
sp_iqprocparm sp_test, DBA	Displays parameters for the procedure **sp_test** owned by `DBA`

Syntax	Output
sp_iqprocparm sp_iqtable	**sp_iqtable** is a system procedure. If there is no user-defined procedure also named **sp_iqtable**, no rows are returned. (By default, only user-defined procedures are returned.)
sp_iqprocparm sp_iqtable, dbo	No rows returned, as the procedure **sp_iqtable** is not a user procedure. (By default, only user procedures are returned.)
sp_iqprocparm NULL, NULL, SYSTEM	Displays parameters for all system procedures (owned by `dbo` or `SYS`)
sp_iqprocparm sp_iqtable, NULL, SYSTEM	Displays parameters of the system procedure **sp_iqtable**
sp_iqprocparm sp_iqtable, dbo, ALL	Displays parameters of the system procedure **sp_iqtable** owned by `dbo`

Description

The **sp_iqprocparm** stored procedure displays information about stored procedure parameters, including result set variables and SQLSTATE/SQLCODE error values. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if *proc-name* is specified, only information about parameters to the specified procedure displays. If *proc-owner* is specified, **sp_iqprocparm** only returns information about parameters to procedures owned by the specified owner. If no parameters are specified, **sp_iqprocparm** displays information about parameters to all the user-defined procedures in the database.

The **sp_iqprocparm** procedure returns information in the following columns:

Table 128. sp_iqprocparm columns

Column name	Description
proc_name	The name of the procedure
proc_owner	The owner of the procedure
parm_name	The name of the parameter
parm_type	The type of parameter is one of the following values: • normal parameter (variable) • result variable: used with procedures that return result sets • SQLSTATE error value • SQLCODE error value

Column name	Description
parm_mode	The mode of the parameter: whether a parameter supplies a value to the procedure, returns a value, does both, or does neither. Parameter mode is one of the following: • in: parameter supplies a value to the procedure • out: parameter returns a value • inout: parameter supplies as well as returns a value • NULL: parameter neither supplies nor returns a value
domain_name	The name of the data type of the parameter as listed in the SYSDOMAIN system table
width	The length of string parameters, the precision of numeric parameters, and the number of bytes of storage for all other data types
scale	The number of digits after the decimal point for numeric data type parameters and zero for all other data types
default	The default value of the parameter, held as a string

Examples

Display information about the parameters of the user-defined procedure `sp_test`:

```
sp_iqprocparm sp_test

proc_name proc_owner parm_name parm_type parm_mode domain_name widt
h scale  default

sp_test   DBA          ID
normal    in        integer     4     0      (NULL)
```

Display information about the parameters of the system procedure **sp_iqshowcompression**:

```
sp_iqprocparm sp_iqshowcompression, dbo, system

proc_name            proc_owner  parm_name    parm_type  parm_mode
domain_name  width  scale  default

sp_iqshowcompression  dbo         @owner_name  normal     in
char         128    0     (NULL)
sp_iqshowcompression  dbo         @table_name  normal     in
char         128    0     (NULL)
sp_iqshowcompression  dbo         @column_name normal     in
char         128    0     (NULL)
sp_iqshowcompression  dbo         Column       result     out
char         128    0     (NULL)
sp_iqshowcompression  dbo         Compression  result     out
char         3      0     (NULL)
```

See also

- *sp_iqcolumn Procedure* on page 447
- *sp_iqconstraint Procedure* on page 455
- *sp_iqdatatype Procedure* on page 463
- *sp_iqevent Procedure* on page 484
- *sp_iqhelp Procedure* on page 489
- *sp_iqindex and sp_iqindex_alt Procedures* on page 496
- *sp_iqpkeys Procedure* on page 536
- *sp_iq_reset_identity Procedure* on page 549
- *sp_iqtable Procedure* on page 575
- *sp_iqview Procedure* on page 590

sp_iqrebuildindex Procedure

Rebuilds column indexes.

To rebuild an index other than the default `FP` index, specify the index name. **sp_iqrebuildindex** behavior is the same regardless of the `FP_NBIT_IQ15_COMPATIBILITY` setting.

Syntax

```
sp_iqrebuildindex (table_name, index_clause)
```

Privileges

User must be a table owner, have INSERT permission on a table to rebuild an index on that table, or have the INSERT ANY TABLE system privilege.

Usage

Parameter	Description
table_name	Partial or fully qualified table name on which the index rebuild process takes place. If the user both owns the table and executes the procedure, a partially qualified name may be used; otherwise, the table name must be fully qualified.

Parameter	Description

index_clause	One or more of the following strings, separated by spaces:
	
	column *column_name [count]*
	
	index *index_name*
	
	Each *column_name* or *index_name* must refer to a column or index on the specified table. If you specify a *column_name* or *index_name* multiple times, the procedure returns an error and no index is rebuilt.
	
	The *count* is a non-negative number that represents the **IQ UNIQUE** value. In a **CREATE TABLE** statement, **IQ UNIQUE (count)** approximates how many distinct values can be in a given column. The number of distinct values affects query speed and storage requirements.
	
	MERGEALL and **RETIER** are keywords specific to `HG` index operations:
	
	`sp_iqrebuildindex ('table name', ' index index name [ MERGEALL | RETIER ] ')`
	
	If `MERGEALL` or `RETIER` are omitted from an operation from an `HG` index , **sp_iqrebuildindex** truncates and recontructs the entire `HG` index from the column data.
	
	`MERGEALL` merges all tiers of a tiered `HG` index and moves the contents into an appropriate tier:
	
	`sp_iqrebuildindex(' table name ', ' index index name MERGEALL ')`
	
	The merge ensures that there is only one active sub-index in a tiered `HG` index. `MERGEALL` operations may improve query access time for a tiered index in cases where there are too many deleted records (as shown by **sp_iqindexmetadata**). `MERGEALL` will only be supported with an index clause and only if the index specified is an `HG` index.
	
	`RETIER` is a keyword specific to `HG` indexes that changes the format of an `HG` index from non-tiered `HG` to tiered `HG`, or tiered `HG` to non-tiered `HG`:
	
	`sp_iqrebuildindex(' table name ', ' index index name RETIER ')`
	
	`RETIER` toggles the format of an `HG` index:
	
	• `RETIER` converts a tiered `HG` index into a single non-tiered `HG` index. Tiering metadata is disabled and only one sub-index is maintained.
	• `RETIER` converts a non-tiered `HG` into a tiered `HG` index, and pushes the single sub-index which contains all the data into an appropriate tier.

Parameter	Description
	`MERGEALL` and `RETIER` will only be supported with an index clause, and only if the index specified is an `HG` index.

You must specify the keywords **column** and **index**. These keywords are not case-sensitive.

Note: This procedure does not support **TEXT** indexes. To rebuild a **TEXT** index you must drop and re-create the index.

Description

If you specify a column name, **sp_iqrebuildindex** rebuilds the default `FP` index for that column; no index name is needed. If you specify the default `FP` index name assigned by SAP Sybase IQ in addition to the column name, **sp_iqrebuildindex** returns an error.

sp_iqrebuildindex rebuilds a **WD** index on a column of data type `LONG VARCHAR` (`CLOB`).

A column with `IQ UNIQUE` *n* value determines whether **sp_iqrebuildindex** rebuilds the column as `Flat FP` or `NBit`. An `IQ UNIQUE` *n* value set to 0 rebuilds the index as a `Flat FP`. An *n* value greater than 0 but less than 2,147,483,647 rebuilds the index as `NBit`. `NBit` columns without an *n* value are rebuilt as `NBit`. **sp_iqrebuildindex** rebuilds an `NBit` column as `NBit`, even if you do not specify a count. If you do specify a count, the *n* value must be greater than the number of unique values already in the index.

If you rebuild a column with a `Flat FP` index, and the column does not include an `IQ UNIQUE` *n* value, **sp_iqrebuildindex** rebuilds the index as `Flat FP` up to the limits defined in the `FP_NBIT_AUTOSIZE_LIMIT` and `FP_NBIT_LOOKUP_MB` options. Specifiying a *n* value for a flat column throws an error if `FP_NBIT_ENFORCE_LIMITS=ON` and the cardinality exceeds the count.

The **sp_iqrebuildindex** default interface allows a user to recreate an entire `HG` index from an existing `FP` index. **sp_iqrebuildindex** re-reads all `FP` index column values and creates the `HG` index. This will, however retain all the metadata regarding tier sizes, continuous load size, etc.

Examples

Rebuilds the default `FP` index on column *Surname*:

```
sp_iqrebuildindex ‘emp1‘, ‘column dept_id‘
```

or:

```
call sp_iqrebuildindex (‘empl1‘, ‘column dept_id‘)
```

Creates a flat `FP` index on column `c1`:

```
CREATE TABLE mytable (c1 int IQ UNIQUE (0))
```

Converts the default `Flat FP` index to an `Nbit` index with an estimated distinct count of 1024:

```
sp_iqrebuildindex ‘mytable‘, ‘column c1 1024‘
```

or:

```
call sp_iqrebuildindex (‘mytable‘, ‘column c1 1024‘)
```

Note: Users can expect to see a temporary performance drop when **sp_iqrebuildindex** runs on a large `HG` index.

See also

- *sp_iqindexfragmentation Procedure* on page 501
- *sp_iqrowdensity Procedure* on page 553

sp_iqrename Procedure

Renames user-created tables, columns, indexes, constraints (unique, primary key, foreign key, and check), stored procedures, and functions.

Syntax

sp_iqrename `object-name, new-name [, object-type ]`

Privileges

User must be the owner of the table, or have ALTER permission on the table, or have either the ALTER ANY OBJECT or ALTER ANY <Object_Type> system privilege. Requires exclusive access to the object.

This procedure can rename table, column, index and constraint. Non-privileged users can rename only objects owned by themselves. Users with ALTER permission on a table can rename that table, its columns, and constraints. Users with the ALTER ANY OBJECT system privilege can rename any object. Users with the ALTER ANY TABLE system privilege can rename any table, column or constraint. Users with the ALTER ANY INDEX system privilege can rename any index, but not tables or columns. User with REFERENCES permission on a table can rename only the indexes of that table.

Usage

Table 129. Parameters

Parameter	Description
object-name	The original name of the user-created object. Optionally, *owner-name* can be specified as part of *object-name* as *owner-name.object-name*, where *owner-name* is the name of the owner of the object being renamed. If *owner-name* is not specified, the user calling **sp_iqrename** is assumed to be the owner of the object. The object is successfully renamed only if the user calling **sp_iqrename** has the required permissions to rename the object. If the object to be renamed is a column, index, or constraint, you must specify the name of the table with which the object is associated. For a column, index, or constraint, *object-name* can be of the form *table-name.object-name* or *owner-name.table-name.object-name*.
new-name	The new name of the object. The name must conform to the rules for identifiers and must be unique for the type of object being renamed.
object-type	An optional parameter that specifies the type of the user-created object being renamed, that is, the type of the object *object-name*. The *object-type* parameter can be specified in either upper or lowercase.

Values for the object-type parameter:

Table 130. sp_iqrename object-type parameter values

object-type parameter	Specifies
column	The object being renamed is a column
index	The object being renamed is an index
constraint	The object being renamed is a unique, primary key, check, or referential (foreign key) constraint

***object-type* parameter**	**Specifies**
procedure	The object being renamed is a function
object-type not specified	The object being renamed is a table

Warning! You must change appropriately the definition of any dependent object (procedures, functions, and views) on an object being renamed by **sp_iqrename**. The **sp_iqrename** procedure does not automatically update the definitions of dependent objects. You must change these definitions manually.

Description

The **sp_iqrename** stored procedure renames user-created tables, columns, indexes, constraints (unique, primary key, foreign key, and check), and functions.

If you attempt to rename an object with a name that is not unique for that type of object, **sp_iqrename** returns the message "`Item already exists.`"

sp_iqrename does not support renaming a view, a procedure, an event or a data type. The message "`Feature not supported.`" is returned by **sp_iqrename**, if you specify **event** or **datatype** as the *object-type* parameter.

You can also rename using the **RENAME** clause of the **ALTER TABLE** statement and **ALTER INDEX** statement.

Examples

Renames the table `titles` owned by user `shweta` to `books`:

```
sp_iqrename shweta.titles, books
```

Renames the column `id` of the table `books` to `isbn`:

```
sp_iqrename shweta.books.id, isbn, column
```

Renames the index `idindex` on the table `books` to `isbnindex`:

```
sp_iqrename books.idindex, isbnindex, index
```

Renames the primary key constraint `prim_id` on the table `books` to `prim_isbn`:

```
sp_iqrename books.prim_id, prim_isbn, constraint
```

sp_iq_reset_identity Procedure

Sets the seed of the Identity/Autoincrement column associated with the specified table to the specified value.

Syntax

```
sp_iq_reset_identity (table_name, table_owner, value)
```

Usage

You must specify *table_name*, *table owner*, and *value*.

Privileges
The user must be the table owner, or have ALTER permission on the table, or have the ALTER ANY TABLE or ALTER ANY OBJECT system privilege.

Description
The Identity/Autoincrement column stores a number that is automatically generated. The values generated are unique identifiers for incoming data. The values are sequential, are generated automatically, and are never reused, even when rows are deleted from the table. The seed value specified replaces the default seed value and persists across database shutdowns and failures.

See also
- *sp_iqcolumn Procedure* on page 447
- *sp_iqconstraint Procedure* on page 455
- *sp_iqdatatype Procedure* on page 463
- *sp_iqevent Procedure* on page 484
- *sp_iqhelp Procedure* on page 489
- *sp_iqindex and sp_iqindex_alt Procedures* on page 496
- *sp_iqpkeys Procedure* on page 536
- *sp_iqprocparm Procedure* on page 540
- *sp_iqtable Procedure* on page 575
- *sp_iqview Procedure* on page 590

sp_iq_reset_identity Procedure Example

Use the example as a reference for **sp_iq_reset_identity** usage.

The following example creates an Identity column with a starting seed of 50:

```
CREATE TABLE mytable(c1 INT identity)
```

```
call sp_iq_reset_identity('mytable', 'dba', 50)
```

sp_iqrestoreaction Procedure

Shows what restore actions are needed to bring database to a consistent state with a given past date.

Syntax

sp_iqrestoreaction '*timestamp*'

Usage

Table 131. Parameters

Parameter	Description
timestamp	Specifies the past date target.

Privileges
No specific system privilege is required to run this procedure.

Description
`sp_iqrestoreaction` returns an error if the database cannot be brought to a consistent state for the timestamp. Otherwise, suggests restore actions that will return the database to a consistent state.

The common point to which the database can be restored coincides with the last backup time that backed up read-write files just before the specified timestamp. The backup may be all-inclusive or read-write files only.

Output may not be in exact ascending order based on backup time. If a backup archive consists of multiple read-only dbfiles, it may contain multiple rows (with the same backup time and backup id).

If you back up a read-only dbfile or dbspace multiple times, the restore uses the last backup. The corresponding backup time could be after the specified timestamp, as long as the dbspace/dbfile alter ID matches the dbspace/dbfile alter ID recorded in the last read-write backup that is restored.

sp_iqrestoreaction returns the following:

Table 132. sp_iqrestoreaction columns

Column name	Description
sequence_number	Orders the steps to be taken
backup_id	Identifier for the backup transaction
backup_archive_list	List of archive files in the backup
backup_time	Time of the backup taken
virtual_type	Type of virtual backup: “Non-virtual,” “Decoupled,” or “Encapsulated”
restore_dbspace	Can be empty. Indicates that all dbspaces are to be restored from the backup archive

Column name	Description
restore_dbfile	Could be empty. Indicates that all dbfiles in the given dbspace are to be restored from the backup archive
backup_comment	User comment

Example

Sample output of **sp_iqrestoreaction**:

```
sequence_number   backup_id   backup_archive_list    backup_time
              1        1192   c:\\\\temp\\\\b1       2008-09-23
14:47:40.0
              2        1201   c:\\\\temp\\\\b2.inc   2008-09-23
14:48:05.01
              3        1208   c:\\\\temp\\\\b3.inc   2008-09-23
14:48:13.0

virtual_type   restore_dbspace   restore_dbfile   backup_comment
Nonvirtual
Nonvirtual
Nonvirtual
```

sp_iqrlvmemory Procedure

Monitors RLV store memory usage per table.

Syntax

```
sp_iqrlvmemory (table_name, [ table_owner ])
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted EXECUTE permission to run the stored procedure.

Usage

Version-specific data, such as version bitmaps and on-demand indexes, are not included in RLV memory accounting. They do not count against the RLV memory limit, and are not reported in sp_iqrlvmemory.

The table owner and table name are optional. Specify the table owner and/or table name to limit the scope.

Description

sp_iqrlvmemory outputs one row per table consuming RLV store memory, with the following output columns:

Column Name	Description
table_id	ID of the table this row represents.

Column Name	Description
fragments	Number of store fragments for this table.
total	Total RLV store memory used by this table.
data	RLV store memory used for the column fragments for this table.
dictionary	RLV store memory used for the dictionaries for this table.
bitmap	RLV store memory used to store table-level bitmaps.

```
sp_iqrlvmemory ‘DBA’, ‘rlv_table1’
```

sp_iqrowdensity Procedure

Reports information about the internal row fragmentation for a table at the FP index level.

Syntax

dbo.sp_iqrowdensity (‘*target* ’)

target: (**table** *table-name* | (**column** *column-name* (...))

Privileges

User must be a table owner or have the MONITOR, MANAGE ANY DBSPACE, CREATE ANY INDEX, ALTER ANY INDEX, CREATE ANY OBJECT, or ALTER ANY OBJECT system privilege. Users without one of these system privileges must be granted `EXECUTE` permission to run the stored procedure.

Usage

Parameter	Description
table-name	Target table *table-name* reports on all columns in the named table.
column-name	Target column *column-name* reports on the named column in the target table. You may specify multiple target columns, but must repeat the keyword each time.

You must specify the keywords **table** and **column**. These keywords are not case-sensitive.

Description

`sp_iqrowdensity` measures row fragmentation at the default index level. Density is the ratio of the minimum number of pages required by an index for existing table rows to the number of pages actually used by the index. This procedure returns density as a number such that 0 < *density* < 1. For example, if an index that requires 8 pages minimum storage occupies 10 pages, its density is .8.

The density reported does not indicate the number of disk pages that may be reclaimed by re-creating or reorganizing the default index.

This procedure displays information about the row density of a column, but does not recommend further action. You must determine whether or not to re-create, reorganize, or rebuild an index.

Example

Reports the row density on column *ID* in table *SalesOrders*:

```
sp_iqrowdensity('column groupo.SalesOrders.ID')
```

Tablename	ColumnName	IndexType	Density
GROUPO.SalesOrders	ID	NBit FP	1.0

See also

- *sp_iqindexfragmentation Procedure* on page 501
- *sp_iqrebuildindex Procedure* on page 544

sp_iqsetcompression Procedure

Sets compression of data in columns of `LONG BINARY` (`BLOB`) and `LONG VARCHAR` (`CLOB`) data types.

Syntax

```
sp_iqsetcompression ( owner, table, column, on_off_flag )
```

Permissions

ALTER ANY TABLE or ALTER ANY OBJECT system privilege required or own table.

Description

sp_iqsetcompression provides control of compression of `LONG BINARY` (`BLOB`) and `LONG VARCHAR` (`CLOB`) data type columns. The compression setting applies only to base tables.

A side effect of **sp_iqsetcompression** is that a **COMMIT** occurs after you change the compression setting.

Table 133. sp_iqsetcompression parameters

Name	Description
owner	Owner of the table for which you are setting compression
table	Table for which you are setting compression
column	Column for which you are setting compression
on_off_flag	Compression setting: ON enables compression, OFF disables compression

Example

Assume this table definition:

```
CREATE TABLE USR.pixTable (picID INT NOT NULL,
picJPG LONG BINARY NOT NULL);
```

To turn off compression on the LOB column `picJPG`, call **sp_iqsetcompression**:

```
CALL sp_iqsetcompression('USR', 'pixTable', 'picJPG',
'OFF') ;
```

This command returns no rows.

sp_iqsharedtempdistrib Procedure

Shows the current shared temp space usage distribution. If run from the coordinator, **sp_iqsharedtempdistrib** displays shared temp space distribution for all nodes. If run from a secondary node, displays shared temp space usage for only that node.

Shared temporary space is reserved for each node in the multiplex on demand. Space is reserved for a node in an allocation unit. Nodes can have multiple allocation units reserved based on their dynamic space demands. Allocation units are leased to allow nodes to use more space as needed and return the space to a global pool when not needed. Allocation units expire when space usage decreases and their lease time ends, or when a server shuts down.

Syntax

```
sp_iqsharedtempdistrib
```

Applies to

Multiplex only.

Privileges

Requires the MANAGE ANY DBSPACE system privilege. Users without the MANAGE ANY DBSPACE system privilege must be granted `EXECUTE` permission.

Description

Column	Data Type	Description
ServerID	unsigned bigint	Server ID of the multiplex server, from SYSIQMPXINFO.
DBSpaceName	char(128)	Name of the dbspace from which space is reserved.
UnitType	char(10)	Type of allocation unit. Valid values are: • Active – currently reserved and in use by the node. • Expired – reserved for the node but in transition back to the global space pool. • Quarantined – reserved for the node but quarantined due to node failure.
VersionID	unsigned bigint	Version ID of the unit. For active units, the version when the unit was reserved for the node. For expired units, the version when the unit was expired. For quarantined units, the version when the unit was quarantined.
NBlocks	unsigned bigint	Number of outstanding blocks in the unit.

sp_iqshowcompression Procedure

Displays compression settings for columns of `LONG BINARY` (`BLOB`) and `LONG VARCHAR` (`CLOB`) data types.

Syntax

```
sp_iqshowcompression ( owner, table, column )
```

Permissions

ALTER ANY TABLE or ALTER ANY OBJECT system privilege required or own table.

Description

Returns the column name and compression setting. Compression setting values are 'ON' (compression enabled) and 'OFF' (compression disabled).

Table 134. sp_iqshowcompression parameters

Name	Description
owner	Owner of the table for which you are setting compression
table	Table for which you are setting compression
column	Column for which you are setting compression

Example

Assume this table definition:

```
CREATE TABLE USR.pixTable (picID INT NOT NULL,
picJPG LONG BINARY NOT NULL);
```

To check the compression status of the columns in the `pixTable` table, call **sp_iqshowcompression**:

```
CALL sp_iqshowcompression('USR', 'pixTable',
'picJPG') ;
```

This command returns one row:

```
'picJPG','ON'
```

sp_iqshowpsexe Procedure

Displays information about the settings of database options that control the priority of tasks and resource usage for connections.

Syntax

sp_iqshowpsexe `[` *`connection-id`* `]`

Privileges

Requires the DROP CONNECTION, MONITOR, or SERVER OPERATOR system privilege. Users without one of these system privileges must be granted `EXECUTE` permission to run the stored procedure.

Usage

Table 135. Parameters

Parameter	Description
connection-id	An integer representing the connection ID. If *connection-id* is specified, **sp_iqshowpsexe** returns information only about the specified connection. If *connection-id* is not specified, **sp_iqshowpsexe** returns information about all connections. If the specified *connection-id* does not exist, **sp_iqshowpsexe** returns no rows.

Description

The **sp_iqshowpsexe** stored procedure displays information about the settings of database options that control the priority of tasks and resource usage for connections, which is useful to database administrators for performance tuning.

Table 136. sp_iqshowpsexe columns

Column name	Description
connectionid	The connection ID
application	Information about the client application that opened the connection. Includes the **AppInfo** connection property information: HOST: the host name of the client machine EXE: the name of the client executable (Windows only) APPINFO: the APPINFO in the client connection string, if specified
userid	Login name of the user that opened the connection
iqgovern_priority	Value of the database option `IQGOVERN_PRIORITY` that assigns a priority to each query waiting in the -iqgovern queue. By default, this option has a value of 2 (MEDIUM). The values 1, 2, and 3 are shown as HIGH, MEDIUM, and LOW, respectively.
max_query_time	Value of the database option `MAX_QUERY_TIME` that sets a limit, so that the optimizer can disallow very long queries. By default, this option is disabled and has a value of 0.

Column name	Description
query_row_limit	Value if the database option `QUERY_ROWS_RETURNED_LIMIT` that sets the row threshold for rejecting queries based on the estimated size of the result set. The default is 0, which means there is no limit.
query_temp_space_limit	Value of the database option `QUERY_TEMP_SPACE_LIMIT` (in MB) that constrains the use of temporary IQ dbspace by user queries. The default value is 2000MB.
max_cursors	Value of the database option `MAX_CURSOR_COUNT` that specifies a resource governor to limit the maximum number of cursors a connection can use at once. The default value is 50. A value of 0 implies no limit.
max_statements	Value of the database option `MAX_STATEMENT_COUNT` that specifies a resource governor to limit the maximum number of prepared statements that a connection can use at once. The default value is 100. A value of 0 implies no limit.

Note: The **AppInfo** property may not be available from Open Client or jConnect applications such as Interactive SQL. If the **AppInfo** property is not available, the `application` column is blank.

Example

Display information about the settings of database options that control the priority of tasks and resource usage for connection ID 2:

```
sp_iqshowpsexe 2

connectionid    application
           2    HOST=GOODGUY-XP;EXE=C:\\Program Files\\Sybase\\
                IQ-16_0\\bin32\\dbisqlg.exe;

userid    iqgovern_priority    max_query_time    query_row_limit
DBA       MEDIUM                            0                  0

query_temp_space_limit    max_statements    max_cursors
                  2000                50            100
```

See also

- *CONNECTION_PROPERTY Function [System]* on page 176
- *sp_iqcontext Procedure* on page 457

sp_iqspaceinfo Procedure

Displays the number of blocks used by each object in the current database and the name of the dbspace in which the object is located.

Syntax

```
sp_iqspaceinfo [ 'main
| [table table-name | index index-name] [...] ']
```

Applies to

Simplex and multiplex.

Privileges

Requires the MANAGE ANY DBSPACE system privilege. Users without the MANAGE ANY DBSPACE system privilege must be granted `EXECUTE` permission.

Description

For the current database, displays the object name, number of blocks used by each object, and the name of the dbspace. **sp_iqspaceinfo** requires no parameters.

The information returned by **sp_iqspaceinfo** is helpful in managing dbspaces.

If run on a multiplex database, the default parameter is **main**, which returns the size of the shared IQ store.

If you supply no parameter, you must have at least one user-created object, such as a table, to receive results.

Example

This output is from the **sp_iqspaceinfo** stored procedure run on the `iqdemo` database. Output for some tables and indexes are removed from this example.

```
             Name                                  NBlocks    dbspace_name

Contacts                                          19       IQ_SYSTEM_MAIN
SalesOrderItems.DBA.ASIQ_IDX_T205_C5_FP            56       IQ_SYSTEM_MAIN
Contacts.DBA.ASIQ_IDX_T206_C10_FP                 55       IQ_SYSTEM_MAIN
Contacts.DBA.ASIQ_IDX_T206_C1_FP                  61       IQ_SYSTEM_MAIN
...
Contacts.DBA.ASIQ_IDX_T206_C9_FP                  55       IQ_SYSTEM_MAIN
Contacts.DBA.ASIQ_IDX_T206_I11_HG                 19       IQ_SYSTEM_MAIN
Customers                                         20       IQ_SYSTEM_MAIN
Customers.DBA.ASIQ_IDX_T207_C1_FP                 61       IQ_SYSTEM_MAIN
Customers.DBA.ASIQ_IDX_T207_C2_FP                 55       IQ_SYSTEM_MAIN
...
Customers.DBA.ASIQ_IDX_T207_I10_HG                19       IQ_SYSTEM_MAIN
...
```

See also

- *sp_iqindexinfo Procedure* on page 503

- *sp_iqdbspace Procedure* on page 467
- *sp_iqdbspaceinfo Procedure* on page 470

sp_iqspaceused Procedure

Shows information about space available and space used in the IQ store, IQ temporary store, RLV store, and IQ global and local shared temporary stores.

Syntax

```
sp_iqspaceused(out mainKB            unsigned bigint,
               out mainKBUsed        unsigned bigint,
               out tempKB            unsigned bigint,
               out tempKBUsed        unsigned bigint,
               out shTempTotalKB     unsigned bigint,
               out shTempTotalKBUsed unsigned bigint,
               out shTempLocalKB     unsigned bigint,
               out shTempLocalKBUsed unsigned bigint,
               out rlvLogKB          unsigned bigint,
               out rlvLogKBUsed      unsigned bigint)
```

Applies to

Simplex and multiplex.

Privileges

Requires the ALTER DATABASE, MANAGE ANY DBSPACE, or MONITOR system privileges. Users without one of these system privileges must be granted `EXECUTE` permission.

Usage

sp_iqspaceused returns several values as unsigned bigint out parameters. This system stored procedure can be called by user-defined stored procedures to determine the amount of main, temporary, and RLV store space in use.

Description

sp_iqspaceused returns a subset of the information provided by **sp_iqstatus**, but allows the user to return the information in SQL variables to be used in calculations.

If run on a multiplex database, this procedure applies to the server on which it runs. Also returns space used on IQ_SHARED_TEMP.

Column Name	Description
mainKB	The total IQ main store space, in kilobytes.
mainKBUsed	The number of kilobytes of IQ main store space used by the database. Secondary multiplex nodes return '(Null)'.

Column Name	Description
tempKB	The total IQ temporary store space, in kilobytes.
tempKBUsed	The number of kilobytes of total IQ temporary store space in use by the database.
shTempTotalKB	The total IQ global shared temporary store space, in kilobytes.
shTempLocalKB	The total IQ local shared temporary store space, in kilobytes.
shTempLocalKBUsed	The number of kilobytes of IQ local shared temporary store space in use by the database.
rlvLogKB	The total RLV store space, in kilobytes.
rlvLogKBUsed	The number of kilobytes of RLV store space in use by the database.

Example

sp_iqspaceused requires seven output parameters. Create a user-defined stored procedure **myspace** that declares the seven output parameters, then calls **sp_iqspaceused**:

```
create or replace procedure dbo.myspace()
begin
     declare mt unsigned bigint;
     declare mu unsigned bigint;
     declare tt unsigned bigint;
     declare tu unsigned bigint;
     declare gt unsigned bigint;
     declare gu unsigned bigint;
     declare lt unsigned bigint;
     declare lu unsigned bigint;
     declare rvlt unsigned bigint;
     declare rvlu unsigned bigint;
    call sp_iqspaceused(mt,mu,tt,tu,gt,gu,lt,lu, rvlt, rvlu);
     select cast(mt/1024 as unsigned bigint) as mainMB,
     cast(mu/1024 as unsigned bigint) as mainusedMB,
     mu*100/mt as mainPerCent,
     cast(tt/1024 as unsigned bigint) as tempMB,
     cast(tu/1024 as unsigned bigint) as tempusedMB,
     tu*100/tt as tempPerCent,
     cast(gt/1024 as unsigned bigint) as shTempTotalKB,
     if gu=0 then 0 else gu*100/gt endif as globalshtempPerCent,
     cast(lt/1024 as unsigned bigint) as shTempLocalMB,
     cast(lu/1024 as unsigned bigint) as shTempLocalKBUsed,
     if lt=0 then 0 else lu*100/lt endif as localshtempPerCent,
     cast(rvlt/1024 as unsigned bigint) as rlvLogKB,
     cast(rvlu/1024 as unsigned bigint) as rlvLogKBUsed;
end
```

To display the output of **sp_iqspaceused**, execute **myspace**:

```
myspace
```

sp_iqstatistics Procedure

Returns serial number, name, description, value, and unit specifier for each available statistic, or a specified statistic.

Syntax

```
sp_iqstatistics [ stat_name ]
```

Usage

Parameter	Description
stat_name	(Optional) `VARCHAR` parameter specifying the name of a statistic.

Privileges

Requires the MANAGE ANY STATISTICS system privilege. Users without MANAGE ANY STATISTICS system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

When **stat_name** is provided, **sp_iqstatistics** returns one row for the given statistic, or zero rows if the name is invalid. When invoked without any parameter, **sp_iqstatistics** returns all statistics.

Result Set

Column name	Data type	Description
stat_num	UNSIGNED INTEGER	Serial number of a statistic
stat_name	VARCHAR(255)	Name of statistic
stat_desc	VARCHAR(255)	Description of statistic
stat_value	LONG VARCHAR	Value of statistic
stat_unit	VARCHAR(128)	Unit specifier

The following statistics may be returned:

stat_num	stat_name	stat_desc	stat_unit
0	CpuTotalTime	Total CPU time in seconds consumed by the SAP Sybase IQ server since last server startup	Second
1	CpuUserTime	CPU user time in seconds consumed by the SAP Sybase IQ server since last server startup	Second
2	CpuSystemTime	CPU system time in seconds consumed by the SAP Sybase IQ server since last server startup	Second
3	ThreadsFree	Number of SAP Sybase IQ threads free	N/A
4	ThreadsInUse	Number of SAP Sybase IQ threads in use	N/A
5	MemoryAllocated	Allocated memory in megabytes	MB
6	MemoryMaxAllocated	Max allocated memory in megabytes	MB
7	MainCacheCurrentSize	Main cache current size in megabytes	MB
8	MainCacheFinds	Main cache total number of lookup requests	N/A
9	MainCacheHits	Main cache total number of hits	N/A
10	MainCachePagesPinned	Main cache number of pages pinned	Page
11	MainCachePagesPinnedPercentage	Percentage of main cache pages pinned	%
12	MainCachePagesDirtyPercentage	Percentage of main cache pages dirtied	%
13	MainCachePagesInUsePercentage	Percentage of main cache pages in use	%
14	TempCacheCurrentSize	Temporary cache current size in megabytes	MB
15	TempCacheFinds	Temporary cache total number of lookup requests	N/A
16	TempCacheHits	Temporary cache total number of hits	N/A
17	TempCachePagesPinned	Temporary cache number of pages pinned	Page

stat_num	stat_name	stat_desc	stat_unit
18	TempCachePagesPinnedPercentage	Percentage of temporary cache pages pinned	%
19	TempCachePagesDirtyPercentage	Percentage of temporary cache pages dirtied	%
20	TempCachePagesInUsePercentage	Percentage of temporary cache pages in use	%
21	MainStoreDiskReads	Number of kilobytes read from main store	KB
22	MainStoreDiskWrites	Number of kilobytes written to main store	KB
23	TempStoreDiskReads	Number of kilobytes read from main store	KB
24	TempStoreDiskWrites	Number of kilobytes written to main store	KB
25	ConnectionsTotalConnections	Total number of connections since server startup	N/A
26	ConnectionsTotalDisonnections	Total number of disconnections since server startup	N/A
27	ConnectionsActive	Number of active connections	N/A
28	OperationsWaiting	Number of operations waiting for SAP Sybase IQ resource governor	N/A
29	OperationsActive	Number of active concurrent operations admitted by SAP Sybase IQ resource governor	N/A
30	OperationsActiveLoadTableStatements	Number of active LOAD TABLE statements	N/A

Examples

Displays a single statistic, the total CPU time:

```
sp_iqstatistics 'CpuTotalTime'
```

Displays all statistics for MainCache%:

```
SELECT * from sp_iqstatistics() WHERE stat_name LIKE 'MainCache%'
```

stat_num	stat_name	stat_desc	stat_value	stat_unit
7	MainCacheCurrentSize	Main cache current size in megabytes	64	mb
8	MainCacheFinds	Main cache total number of lookup requests	95303	
9	MainCacheHits	Main cache total number of hits	95283	
10	MainCachePagesPinned	Main cache number of pages pinned	0	page
11	MainCachePagesPinnedPercentage	Percentage of main cache pages pinned	0	%
12	MainCachePagesDirtyPercentage	Percentage of main cache pages dirtied	0.39	%
13	MainCachePagesInUsePercentage	Percentage of main cache pages in use	4.44	%

sp_iqstatus Procedure

Displays a variety of SAP Sybase IQ status information about the current database.

Syntax

```
sp_iqstatus
```

Applies to

Simplex and multiplex.

Privileges

Requires the ALTER DATABASE, MANAGE ANY DBSPACE, MONITOR, or SERVER OPERATOR system privilege. Users without one of these system privileges must be granted `EXECUTE` permission.

Description

Shows status information about the current database, including the database name, creation date, page size, number of dbspace segments, block usage, buffer usage, I/O, backup information, and so on.

sp_iqstatus displays an out-of-space status for main and temporary stores. If a store runs into an out-of-space condition, **sp_iqstatus** shows `Y` in the store's out-of-space status display value.

Memory used by the row-level versioning (RLV) store can be monitored with **sp_iqstatus**. The **RLV memory limit** row displays the memory limit as specified by the `-iqrlvmem` server option, or the sa_server_option `rlv_memory_mb`. The `RLV memory used` row displays the amount of memory used by the RLV store.

sp_iqspaceused returns a subset of the same information as provided by **sp_iqstatus**, but allows the user to return the information in SQL variables to be used in calculations.

To display space that can be reclaimed by dropping connections, use **sp_iqstatus** and add the results from the two returned rows:

```
(DBA)> select * from sp_iqstatus() where name like '%Versions:%'
Execution time: 6.25 seconds
Name                 Value
----------------------------
Other Versions: 2 = 1968Mb
Active Txn Versions: 1 = C:2175Mb/D:2850Mb

(First 2 rows)
```

The above example output shows that one active write transaction created 2175MB and destroyed 2850 MB of data. The total data consumed in transactions and not yet released is 4818MB, or 1968MB + 2850MB = 4818MB.

sp_iqstatus omits blocks that will be deallocated at the next checkpoint. These blocks do however, appear in **sp_iqdbspace** output as type X.

In a multiplex, this procedure also lists information about the shared IQ store and IQ temporary store. If **sp_iqstatus**shows a high percentage of main blocks in use on a multiplex server, run **sp_iqversionuse** to see which versions are being used and the amount of space that can be recovered by releasing versions.

Example

Note: This example includes a sample user dbspace named `iq_main` that may not be present in your own databases.

The following output is from the **sp_iqstatus** stored procedure:

```
SAP Sybase IQ (TM)              Copyright (c) 1992-2013 by Sybase,
Inc.
```

```
                                         All rights reserved.
Version:                                 16.0.0.160/120507/D/ELAN/
Sun_x64/OS 5.10/
                                         64bit/2012-05-07 17:36:36
Time Now:                                2013-05-16 09:53:13.590
Build Time:                              2013-05-07 17:36:36
File Format:                             23 on 03/18/1999
Server mode:                         IQ Multiplex Coordinator Server
Catalog Format:                          2
Stored Procedure Revision:               1
Page Size:                               131072/8192blksz/16bpp
Number of Main DB Files :                3
Main Store Out Of Space:                 N
Number of Shared Temp DB Files:          0
Shared Temp Store Out Of Space:          N
Number of Local Temp DB Files :          1
Local Temp Store Out Of Space:           N
DB Blocks: 1-640000                      IQ_SYSTEM_MAIN
DB Blocks: 1045440-130101439             iqmain1
DB Blocks: 2090880-2346879               iqmain2
Local Temp Blocks: 1-384000              IQ_SYSTEM_TEMP
Create Time:                             2013-05-08 15:54:15.549
Update Time:                             2013-05-16 09:53:00.077
Local Temp Blocks:  1-1600               IQ_SYSTEM_TEMP
Create Time:                             2013-05-08 15:54:15.549
Update Time:                             2013-05-16 09:53:00.077
Main IQ Buffers:                         510, 64Mb
Temporary IQ Buffers:                    510, 64Mb
Main IQ Blocks Used:                 157379 of 1126400, 13%=1229Mb,
Max Block#: 2128363
Shared Temporary IQ Blocks Used:       0 of 0, 0%=0Mb, Max Block#: 0
Local Temporary IQ Blocks Used:          81 of 358400, 0%=0Mb, Max
Block#: 81
Main Reserved Blocks Available:          25600 of 25600, 100%=200Mb
Shared Temporary Reserved Blocks Available:  0 of 0, 0%=0Mb
Local Temporary Reserved Blocks Available:   25600 of 25600,
100%=200Mb
IQ Dynamic Memory:                       Current: 178mb, Max: 178mb
Main IQ Buffers:                         Used: 99, Locked: 0
Temporary IQ Buffers:                    Used: 5, Locked: 0
Main IQ I/O:                          I: L60904/P29 O: C5463/D11343/
P9486
                                         D:5450 C:51.3
Temporary IQ I/O:                    I: L12526/P0 O: C165/D319/P157
D:160 C:100.0
Other Versions:                          6 = 0Mb
Active Txn Versions:                     0 = C:0Mb/D:0Mb
Last Full Backup ID:                     0
Last Full Backup Time:
Last Backup ID:
Last Backup Type:                        None
Last Backup Time:
DB Updated:                              1
Blocks in next ISF Backup:               0 Blocks: =0Mb
Blocks in next ISI Backup:               0 Blocks: =0Mb
Main Tlvlog Size:                     Pages: 1, Recs: 193, Replays:
```

```
0/0
DB File Encryption Status:              OFF
```

The following is a key to understanding the `Main IQ I/O` and `Temporary IQ I/O` output codes:

- `I:` Input
- `L:` Logical pages read (“Finds”)
- `P:` Physical pages read
- `O:` Output
- `C:` Pages created
- `D:` Pages dirtied
- `P:` Physically written
- `D:` Pages destroyed
- `C:` Compression ratio

See also

- *sp_iqtransaction Procedure* on page 581
- *sp_iqversionuse Procedure* on page 588

sp_iqsysmon Procedure

Monitors multiple components of SAP Sybase IQ, including the management of buffer cache, memory, threads, locks, I/O functions, and CPU utilization.

Batch Mode Syntax

```
sp_iqsysmon start_monitor
sp_iqsysmon stop_monitor [, “section(s)” ]
or
sp_iqsysmon “time-period” [, “section(s)” ]
```

File Mode Syntax

```
sp_iqsysmon start_monitor, ‘filemode’ [, ”monitor-options” ]
sp_iqsysmon stop_monitor
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Batch Mode Usage

Parameter	Description
start_monitor	Starts monitoring.

Parameter	Description
stop_monitor	Stops monitoring and displays the report.
time-period	The time period for monitoring, in the form HH:MM:SS.
section(s)	The abbreviation for one or more sections to be shown by **sp_iqsysmon**. If you specify more than one section, separate the section abbreviations using spaces, and enclose the list in single or double quotes. The default is to display all sections. For sections related to the IQ store, you can specify main or temporary store by prefixing the section abbreviation with "m" or "t", respectively. Without the prefix, both stores are monitored. For example, if you specify "mbufman", only the IQ main store buffer manager is monitored. If you specify "mbufman tbufman" or "bufman", both the main and temporary store buffer managers are monitored.

Report Section or IQ Component	Abbreviation
Buffer manager	(m/t)bufman
Buffer pool	(m/t)bufpool
Prefetch management	(m/t)prefetch
Free list management	(m/t)freelist
Buffer allocation	(m/t)bufalloc
Memory management	memory
Thread management	threads
CPU utilization	cpu
Transaction management	txn
Server context statistics	server
Catalog statistics	catalog

Note: The SAP Sybase IQ components Disk I/O and Lock Manager are not currently supported by **sp_iqsysmon**.

File Mode Usage

Parameter	Description
start_monitor	Starts monitoring.
stop_monitor	Stops monitoring and writes the remaining output to the log file.
filemode	Specifies that **sp_iqsysmon** is running in file mode. In file mode, a sample of statistics appear for every interval in the monitoring period. By default, the output is written to a log file named *dbname.connid-iqmon*. Use the **file_suffix** option to change the suffix of the output file. See the *monitor_options* parameter for a description of the **file_suffix** option.
monitor_options	The *monitor _options* string

The *monitor _options* string can include one or more options:

Table 137. monitor_options string options

monitor_options String Option	Description
-interval *seconds*	Specifies the reporting interval, in seconds. A sample of monitor statistics is output to the log file after every interval. The default is every 60 seconds, if the **-interval** option is not specified. The minimum reporting interval is 2 seconds. If the interval specified for this option is invalid or less than 2 seconds, the interval is set to 2 seconds. The first display shows the counters from the start of the server. Subsequent displays show the difference from the previous display. You can usually obtain useful results by running the monitor at the default interval of 60 seconds during a query with performance problems or during a time of day that generally has performance problems. A very short interval may not provide meaningful results. The interval should be proportional to the job time; 60 seconds is usually more than enough time.

monitor_options String Option	Description
-file_suffix *suffix*	Creates a monitor output file named `dbname.connid-suffix`. If you do not specify the **-file_suffix** option, the suffix defaults to `iqmon`. If you specify the **-file_suffix** option and do not provide a suffix or provide a blank string as a suffix, no suffix is used.
-append or **-truncate**	Directs **sp_iqsysmon** to append to the existing output file or truncate the existing output file, respectively. Truncate is the default. If both options are specified, the option specified later in the string takes precedence.
-section *section(s)*	Specifies the abbreviation of one or more sections to write to the monitor log file. The default is to write all sections. The abbreviations specified in the sections list in file mode are the same abbreviations used in batch mode. When more than one section is specified, spaces must separate the section abbreviations. If the **-section** option is specified with no sections, none of the sections are monitored. An invalid section abbreviation is ignored and a warning is written to the IQ message file.

Usage Syntax Examples

Syntax	Result
sp_iqsysmon start_monitor sp_iqsysmon stop_monitor	Starts the monitor in batch mode and displays all sections for the main and temporary stores
sp_iqsysmon start_monitor sp_iqsysmon stop_monitor “mbufman mbufpool”	Starts the monitor in batch mode and displays the Buffer Manager and Buffer Pool statistics for themain store
sp_iqsysmon “00:00:10”, “mbufpool memory”	Runs the monitor in batch mode for 10 seconds and displays the consolidated statistics at the end of the time period
sp_iqsysmon start_monitor, ‘filemode’, “-interval 5 -sections mbufpool memory” sp_iqsysmon stop_monitor	Starts the monitor in file mode and writes statistics for Main Buffer Pool and Memory Manager to the log file every 5 seconds

Description

The **sp_iqsysmon** stored procedure monitors multiple components of SAP Sybase IQ, including the management of buffer cache, memory, threads, locks, I/O functions, and CPU utilization.

The **sp_iqsysmon** procedure supports two modes of monitoring:

- Batch mode –

  In batch mode, **sp_iqsysmon** collects the monitor statistics for the period between starting and stopping the monitor or for the time period specified in the *time-period* parameter. At the end of the monitoring period, **sp_iqsysmon** displays a list of consolidated statistics.

  sp_iqsysmon in batch mode is similar to the SAP® Sybase Adaptive Server Enterprise procedure **sp_sysmon**.
- File mode –

  In file mode, **sp_iqsysmon** writes the sample statistics in a log file for every interval period between starting and stopping the monitor.

  The first display in file mode shows the counters from the start of the server. Subsequent displays show the difference from the previous display.

  sp_iqsysmon in file mode is similar to the **IQ UTILITIES** command **START MONITOR** and **STOP MONITOR** interface.

Batch Mode Examples

Prints monitor information after 10 minutes:

```
sp_iqsysmon “00:10:00”
```

Prints only the Memory Manager section of the **sp_iqsysmon** report after 5 minutes:

```
sp_iqsysmon “00:05:00”, memory
```

Starts the monitor, executes two procedures and a query, stops the monitor, then prints only the Buffer Manager section of the report:

```
sp_iqsysmon start_monitor
go
execute proc1
go
execute proc2
go
select sum(total_sales) from titles
go
sp_iqsysmon stop_monitor, bufman
go
```

Prints only the Main Buffer Manager and Main Buffer Pool sections of the report after 20 minutes:

```
sp_iqsysmon “00:02:00”, “mbufman mbufpool”
```

File Mode Examples

Truncates and writes information to the log file every 2 seconds between starting the monitor and stopping the monitor:

```
sp_iqsysmon start_monitor, ‘filemode’, ‘-interval 2’
.
.
.
sp_iqsysmon stop_monitor
```

Appends output for only the Main Buffer Manager and Memory Manager sections to an ASCII file with the name `dbname.connid-testmon`. For the database `iqdemo`, writes results in the file `iqdemo.2-testmon`:

```
sp_iqsysmon start_monitor, ‘filemode’,
“-file_suffix testmon -append -section mbufman memory”
.
.
.
sp_iqsysmon stop_monitor
```

Example

Run the monitor in batch mode for 10 seconds and display the consolidated statistics at the end of the time period

```
sp_iqsysmon “00:00:10”, “mbufpool memory”

==============================
Buffer Pool (Main)
==============================
STATS-
NAME          TOTAL  NONE  BTREEV  BTREEF  BV  VDO  DBEXT  DBID  SORT
MovedToMRU         0      0     0       0      0   0     0      0     0
MovedToWash        0      0     0       0      0   0     0      0     0
RemovedFromLRU     0      0     0       0      0   0     0      0     0
RemovedFromWash    0      0     0       0      0   0     0      0     0
RemovedInScanMode  0      0     0       0      0   0     0      0     0

STORE  GARRAY  BARRAY  BLKMAP  HASH  CKPT  BM  TEST  CMID  RIDCA  LOB
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0
0      0       0       0       0     0     0   0     0     0      0

STATS-NAME                     VALUE
Pages                          127   ( 100.0 %)
InUse                          4     ( 3.1 %)
Dirty                          1     ( 0.8 %)
Pinned                         0     ( 0.0 %)
Flushes                        0
FlushedBufferCount             0
GetPageFrame                   0
GetPageFrameFailure            0
```

```
GotEmptyFrame                  0
Washed                         0
TimesSweepersWoken             0

washTeamSize                   0
WashMaxSize                    26   ( 20.5 %)
washNBuffers                   4     ( 3.1 %)
washNDirtyBuffers              1     ( 0.8
%)
                   washSignalThreshold            3      ( 2.4 %)
washNActiveSweepers            0
washIntensity                  1

==============================
Memory Manager
==============================
STATS-NAME                   VALUE
MemAllocated                 43616536     (  42594 KB)
MemAllocatedMax              43735080     (  42710 KB)
MemAllocatedEver             0            (      0 KB)
MemNAllocated                67079
MemNAllocatedEver            0
MemNTimesLocked              0
MemNTimesWaited              0            (   0.0 %)
```

sp_iqtable Procedure

Displays information about tables in the database.

Syntax1

```
sp_iqtable ( [ table_name ],[table_owner ],[table_type ] )
```

Syntax2

```
sp_iqtable [table_name='tablename'],
[table_owner='tableowner' ],[table_type='tabletype' ]
```

Privileges

No specific system privileges are required to run the procedure.

Usage: Syntax1

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqtable NULL,NULL,TEMP` and `sp_iqtable NULL,dbo,SYSTEM`.

Note: The *table_type* values ALL and VIEW must be enclosed in single quotes in Syntax1.

Usage: Syntax2

The parameters can be specified in any order. Enclose them in single quotes.

The allowed values for the *table_type* parameter:

table_type value	Information displayed
TEMP	Global temporary tables
VIEW	Views
ALL	IQ tables, global temporary tables, and views
any other value	IQ tables

Description

Specifying one parameter returns only the tables that match that parameter. Specifying more than one parameter filters the results by all of the parameters specified. Specifying no parameters returns all SAP Sybase IQ tables in the database. There is no method for returning the names of local temporary tables.

Column name	Description
table_name	The name of the table.
table_type	BASE – a base table. MAT VIEW - a materialized view. (SA tables only) GBL TEMP - a global temporary table. PARTITION - a table partition (this table is for internal use only and cannot be used by SAP Sybase IQ users). VIEW – a view.
table_owner	The owner of the table
server_type	IQ – an object created in the IQ store. SA – an object created in the SA store. All views are created in the SA store.
location	TEMP – IQ temporary store. MAIN – IQ store. SYSTEM – catalog store.
dbspace_id	Number that identifies the dbspace.
isPartitioned	'Y' if the column belongs to a partitioned table and has one or more partitions whose dbspace is different from the table partition's dbspace, 'N' if the column's table is not partitioned or each partition of the column resides in the same dbspace as the table partition.
remarks	User comments added with the **COMMENT** statement.

Column name	Description
table_constraints	Constraints against the table.
isRLV	Indicates if the table is RLV-enabled.

See also

- *sp_iqcolumn Procedure* on page 447
- *sp_iqconstraint Procedure* on page 455
- *sp_iqdatatype Procedure* on page 463
- *sp_iqevent Procedure* on page 484
- *sp_iqhelp Procedure* on page 489
- *sp_iqindex and sp_iqindex_alt Procedures* on page 496
- *sp_iqpkeys Procedure* on page 536
- *sp_iqprocparm Procedure* on page 540
- *sp_iq_reset_identity Procedure* on page 549
- *sp_iqview Procedure* on page 590

sp_iqtable Procedure Examples

sp_iqtable output examples.

The following variations in syntax both return information about the table `Departments`:

```
sp_iqtable ('Departments')
```

```
sp_iqtable table_name='Departments'
```

Table_name	Table_type	Table_owner
Departments	BASE	GROUPO

Server_type	location	dbspace_id
IQ	Main	16387

isPartitioned	Remarks	table_constraints
N	contains the names and heads of the various departments in the sporting goods company	(NULL)

The following variations in syntax both return all tables that are owned by table owner `GROUPO`:

```
sp_iqtable NULL,GROUPO
```

```
sp_iqtable table_owner='GROUPO'
```

Table_name	Table_type	Table_owner	Server_type	location
Contacts	BASE	GROUPO	IQ	Main
Customers	BASE	GROUPO	IQ	Main
Departments	BASE	GROUPO	IQ	Main
Employees	BASE	GROUPO	IQ	Main
FinancialCodes	BASE	GROUPO	IQ	Main
FinancialData	BASE	GROUPO	IQ	Main
Products	BASE	GROUPO	IQ	Main
SalesOrders	BASE	GROUPO	IQ	Main
SalesOrderItems	BASE	GROUPO	IQ	Main

dbspace_id	isPartitioned	Remarks	table_constraints
16387	N	names, addresses, and telephone numbers of all people with whom the company wishes to retain contact information	(NULL)
16387	N	customers of the sporting goods company	(NULL)
16387	N	contains the names and heads of the various departments in the sporting goods company	(NULL)
16387	N	contains information such as names, salary, hire date and birthday	(NULL)
16387	N	types of revenue and expenses that the sporting goods company has	(NULL)
16387	N	revenues and expenses of the sporting goods company	(NULL)
16387	N	products sold by the sporting goods company	(NULL)
16387	N	individual items that make up the sales orders	(NULL)
16387	N	sales orders that customers have submitted to the sporting goods company	(NULL)

sp_iqtablesize Procedure

Returns the size of the specified table.

Syntax

```
sp_iqtablesize ( table_owner.table_name )
```

Privileges

User must be a table owner or have the MANAGE ANY DBSPACE or ALTER ANY TABLE system privilege. Users without one of these system privileges must be granted `EXECUTE` permission to run the stored procedure.

Description

Returns the total size of the table in KBytes and NBlocks (IQ blocks). Also returns the number of pages required to hold the table in memory, and the number of IQ pages that are compressed when the table is compressed (on disk). You must specify the *table_name* parameter with this procedure. If you are the owner of *table_name*, then you do not have to specify the *table_owner* parameter.

Column name	Description
Ownername	Name of owner
Tablename	Name of table
Columns	Number of columns in the table
KBytes	Physical table size in KB
Pages	Number of IQ pages needed to hold the table in memory
CompressedPages	Number of IQ pages that are compressed, when the table is compressed (on disk)
NBlocks	Number of IQ blocks
RlvLogPages	Number of IQ pages needed to hold the RLV table log information on disk
RlvLogKBytes	Size of the RLV table log, in kilobytes.

`Pages` is the total number of IQ pages for the table. The unit of measurement for pages is IQ page size. All in-memory buffers (buffers in the IQ buffer cache) are the same size.

IQ pages on disk are compressed. Each IQ page on disk uses 1 to 16 blocks. If the IQ page size is 128KB, then the IQ block size is 8KB. In this case, an individual on-disk page could be 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, or 128 KB.

If you divide the `KBytes` value by page size, you see the average on-disk page size.

Note: SAP Sybase IQ always reads and writes an entire page, not blocks. For example, if an individual page compresses to 88K, then IQ reads and writes the 88K in one I/O. The average page is compressed by a factor of 2 to 3.

`NBlocks` is `Kbytes` divided by IQ block size.

`CompressedPages` is the number of pages that are compressed. For example, if `Pages` is 1000 and `CompressedPages` is 992, this means that 992 of the 1000 pages are compressed. `CompressedPages` divided by `Pages` is usually near 100%, because most pages compress. An empty page is not compressed, since SAP Sybase IQ does not write empty pages. IQ pages compress well, regardless of the fullness of the page.

Example

```
call sp_iqtablesize ('dba.t1')
```

Ownername	Tablename	Columns	KBytes	Pages	CompressedPages	NBlocks	RlvLogPages	RlvLogBytes
DBA	t1	3	192	5	4	24	96	12288

sp_iqtableuse Procedure

Reports detailed usage information for tables accessed by the workload.

Syntax

```
sp_iqtableuse
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

Tables created in SYSTEM are not reported.

Table 138. sp_iqtableuse columns

Column name	Description
TableName	Table name
Owner	User name of table owner
UID**	Table unique identifier

Column name	Description
LastDT	Date/time of last access
NRef	Number of query references

**UID is a number assigned by the system that uniquely identifies the instance of the table (where instance is defined when an object is created).

See also

- *sp_iqcolumnuse Procedure* on page 450
- *sp_iqindexadvice Procedure* on page 499
- *sp_iqindexuse Procedure* on page 512
- *sp_iqunusedcolumn Procedure* on page 585
- *sp_iqunusedindex Procedure* on page 586
- *sp_iqunusedtable Procedure* on page 587
- *sp_iqworkmon Procedure* on page 596

sp_iqtransaction Procedure

Shows information about transactions and versions.

Syntax

```
sp_iqtransaction
```

Applies to

Simplex and multiplex.

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

sp_iqtransaction returns a row for each transaction control block in the SAP Sybase IQ transaction manager. The columns Name, Userid, and ConnHandle are the connection properties **Name**, **Userid**, and **Number**, respectively. Rows are ordered by TxnID.

sp_iqtransaction output does not include connections without transactions in progress. To include all connections, use **sp_iqconnection**.

Note: Although you can use **sp_iqtransaction** to identify users who are blocking other users from writing to a table, **sp_iqlocks** is a better choice for this purpose.

Column Name	Description
Name	The name of the application.
Userid	The user ID for the connection.
TxnID	The transaction ID of this transaction control block. The transaction ID is assigned during **begin transaction**. It appears in the `.iqmsg` file by the BeginTxn, CmtTxn, and PostCmtTxn messages, and is the same as the Txn ID Seq that is logged when the database is opened.
CmtID	The ID assigned by the transaction manager when the transaction commits. For active transactions, the CmtID is zero.
VersionID	For simplex and multiplex nodes, a value of 0 indicates that the transaction is unversioned, and the VersionID has not been assigned. For the multiplex coordinator, the VersionID is assigned after the transaction establishes table locks. Multiplex secondary servers receive the VersionID from the coordinator. The VersionID is used internally by the SAP Sybase IQ in-memory catalog and the IQ transaction manager to uniquely identify a database version to all nodes within a multiplex database.
State	The state of the transaction control block. This variable reflects internal SAP Sybase IQ implementation details and is subject to change in the future. Currently, transaction states are `NONE`, `ACTIVE`, `ROLLING_BACK`, `ROLLED_BACK`, `COMMITTING`, `COMMITTED`, and `APPLIED`. `NONE`, `ROLLING_BACK`, `ROLLED_BACK`, `COMMITTING` and `APPLIED` are transient states with a very small life span. `ACTIVE` indicates that the transaction is active. `COMMITTED` indicates that the transaction has completed and is waiting to be `APPLIED`, at which point a version that is invisible to any transaction is subject to garbage collection. Once the transaction state is `ROLLED_BACK`, `COMMITTED`, or `APPLIED`, ceases to own any locks other than those held by open cursors.
ConnHandle	The ID number of the connection.
IQConnID	The ten-digit connection ID that is included as part of all messages in the `.iqmsg` file. This is a monotonically increasing integer unique within a server session.
MainTableKBCr	The number of kilobytes of IQ store space created by this transaction.

Column Name	Description
MainTableKBDr	The number of kilobytes of IQ store space dropped by this transaction, but which persist on disk in the store because the space is visible in other database versions or other savepoints of this transaction.
TempTableKBCr	The number of kilobytes of IQ temporary store space created by this transaction for storage of IQ temporary table data.
TempTableKBDr	The number of kilobytes of IQ temporary table space dropped by this transaction, but which persist on disk in the IQ temporary store because the space is visible to IQ cursors or is owned by other savepoints of this transaction.
TempWorkSpaceKB	For ACTIVE transactions, a snapshot of the work space in use at this instant by this transaction, such as sorts, hashes, and temporary bitmaps. The number varies depending on when you run **sp_iqtransaction**. For example, the query engine might create 60MB in the temporary cache but release most of it quickly, even though query processing continues. If you run **sp_iqtransaction** after the query finishes, this column shows a much smaller number. When the transaction is no longer active, this column is zero. For ACTIVE transactions, this column is the same as the TempWorkSpaceKB column of **sp_iqconnection**.
TxnCreateTime	The time the transaction began. All SAP Sybase IQ transactions begin implicitly as soon as an active connection is established or when the previous transaction commits or rolls back.
CursorCount	The number of open SAP Sybase IQ cursors that reference this transaction control block. If the transaction is ACTIVE, it indicates the number of open cursors created within the transaction. If the transaction is COMMITTED, it indicates the number of hold cursors that reference a database version owned by this transaction control block.
SpCount	The number of savepoint structures that exist within the transaction control block. Savepoints may be created and released implicitly. Therefore, this number does not indicate the number of user-created savepoints within the transaction.
SpNumber	The active savepoint number of the transaction. This is an implementation detail and might not reflect a user-created savepoint.
MPXServerName	Indicates if an active transaction is from an internode communication (INC) connection. If from INC connection, the value is the name of the multiplex server where the transaction originates. NULL if not from an INC connection. Always NULL if the transaction is not active.

Column Name	Description
GlobalTxnID	The global transaction ID associated with the current transaction, 0 (zero) if none.
VersioningType	The snapshot versioning type of the transaction; either table-level (the default), or row-level. Row-level snapshot versioning (RLV) applies only to RLV-enabled tables. Once a transaction is started, this value cannot change.
Blocking	Indicates if connection blocking is enabled (True) or disabled (False). You set connection blocking using the **BLOCKING** database option. If true, the transaction blocks, meaning it waits for a conflicting lock to release before it attempts to retry the lock request.
BlockingTimeout	Indicates the time, in milliseconds, a transaction waits for a locking conflict to clear. You set the timeout threshold using the **BLOCKING_TIMEOUT** database option. A value of 0 (default) indicates that the transaction waits indefinitely.

Example

Example **sp_iqtransaction** output:

```
Name   Userid  TxnID  CmtID VersionID   State    ConnHandle  IQConnID
====== ====== ====== ====== =========  ========== ===========
========
red2      DBA  10058  10700     10058  Active     419740283       14


MainTableKBCr      MainTableKBDr    TempTableKBCr TempTableKBDr
============= ================== ================ =============
          0                  0            65824             0


TempWorkSpaceKB TxnCreateTime              CursorCount SpCount
SpNumber
==============
=======================     =========== ======= ========
      0       2013-03-26 13:17:27.612            1       3        2


MPXServerName  GlobalTxnID   VersioningType  Blocking
BlockingTimeout
=============  ===========   ==============  ========
===============
        (NULL)         0     Row-level       True
0
```

See also

- *sp_iqstatus Procedure* on page 566
- *sp_iqversionuse Procedure* on page 588

sp_iqunusedcolumn Procedure

Reports IQ columns that were not referenced by the workload.

Syntax

```
sp_iqunusedcolumn
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

Columns from tables created in SYSTEM or local temporary tables are not reported.

Table 139. sp_iqunusedcolumn columns

Column name	Description
TableName	Table name
ColumnName	Column name
Owner	User name of column owner

Example

Sample output from the **sp_iqunusedcolumn** procedure:

```
TableName                ColumnName               Owner
SalesOrders              ID                       GROUPO
SalesOrders              CustomerID               GROUPO
SalesOrders              OrderDate                GROUPO
SalesOrders              FinancialCode            GROUPO
SalesOrders              Region                   GROUPO
SalesOrders              SalesRepresentative      GROUPO
SalesOrderItems          ID                       GROUPO
SalesOrderItems          LineID                   GROUPO
SalesOrderItems          ProductID                GROUPO
SalesOrderItems          Quantity                 GROUPO
SalesOrderItems          ShipDate                 GROUPO
Contacts                 ID                       GROUPO
Contacts                 Surname                  GROUPO
Contacts                 GivenName                GROUPO ...
```

See also

- *sp_iqcolumnuse Procedure* on page 450
- *sp_iqindexadvice Procedure* on page 499

- *sp_iqindexuse Procedure* on page 512
- *sp_iqtableuse Procedure* on page 580
- *sp_iqunusedindex Procedure* on page 586
- *sp_iqunusedtable Procedure* on page 587
- *sp_iqworkmon Procedure* on page 596

sp_iqunusedindex Procedure

Reports IQ secondary (non-FP) indexes that were not referenced by the workload.

Syntax

```
sp_iqunusedindex
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

Indexes from tables created in SYSTEM or local temporary tables are not reported.

Table 140. sp_iqunusedindex columns

Column name	Description
IndexName	Index name
TableName	Table name
Owner	User name of index owner
IndexType	Index type

Example

Sample output from the **sp_iqunusedindex** procedure:

```
IndexName            TableName              Owner  IndexType
ASIQ_IDX_T450_I7_HG  SalesOrders            GROUPO HG
ASIQ_IDX_T450_C6_HG  SalesOrders            GROUPO HG
ASIQ_IDX_T450_C4_HG  SalesOrders            GROUPO HG
ASIQ_IDX_T450_C2_HG  SalesOrders            GROUPO HG
ASIQ_IDX_T451_I6_HG  SalesOrderItems        GROUPO HG
ASIQ_IDX_T451_C3_HG  SalesOrderItems        GROUPO HG
ASIQ_IDX_T451_C1_HG  SalesOrderItems        GROUPO HG
ASIQ_IDX_T452_I11_HG Contacts               GROUPO HG
ASIQ_IDX_T453_I10_HG Contacts               GROUPO HG
ASIQ_IDX_T454_I4_HG  FinancialCodes         GROUPO HG
ASIQ_IDX_T455_I5_HG  FinancialData          GROUPO HG
```

```
ASIQ_IDX_T455_C3_HG  FinancialData          GROUPO HG
ASIQ_IDX_T456_I8_HG  Products               GROUPO HG
ASIQ_IDX_T457_I4_HG  Departments            GROUPO HG
ASIQ_IDX_T457_C3_HG  Departments            GROUPO HG
ASIQ_IDX_T458_I21_HG Departments            GROUPO HG
ASIQ_IDX_T458_C5_HG  Departments            GROUPO HG
```

See also

- *sp_iqcolumnuse Procedure* on page 450
- *sp_iqindexadvice Procedure* on page 499
- *sp_iqindexuse Procedure* on page 512
- *sp_iqtableuse Procedure* on page 580
- *sp_iqunusedcolumn Procedure* on page 585
- *sp_iqunusedtable Procedure* on page 587
- *sp_iqworkmon Procedure* on page 596

sp_iqunusedtable Procedure

Reports IQ tables that were not referenced by the workload.

Syntax

```
sp_iqunusedtable
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

Tables created in SYSTEM and local temporary tables are not reported.

Table 141. sp_iqunusedtable columns

Column name	Description
TableName	Table name
Owner	User name of table owner

Example

The following table illustrates sample output from the **sp_iqunusedtable** procedure.

```
TableName                Owner FinancialCodes          GROUPO
Contacts                 GROUPO Employees               GROUPO
emp1                     DBA SalesOrders             GROUPO
FinancialData            GROUPO Departments             GROUPO
SalesOrderItems          GROUPO Products                GROUP
```

```
iq_dummy                    DBA Customers                    GROUPO
sale                        DBA
```

See also

- *sp_iqcolumnuse Procedure* on page 450
- *sp_iqindexadvice Procedure* on page 499
- *sp_iqindexuse Procedure* on page 512
- *sp_iqtableuse Procedure* on page 580
- *sp_iqunusedcolumn Procedure* on page 585
- *sp_iqunusedindex Procedure* on page 586
- *sp_iqworkmon Procedure* on page 596

sp_iqversionuse Procedure

Displays version usage for the IQ main store.

Syntax

```
sp_iqversionuse
```

Applies to

Simplex and multiplex.

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Description

The **sp_iqversionuse** system stored procedure helps troubleshoot situations where the databases uses excessive storage space due to multiple table versions.

If out-of-space conditions occur or **sp_iqstatus** shows a high percentage of main blocks in use on a multiplex server, run **sp_iqversionuse** to find out which versions are being used and the amount of space that can be recovered by releasing versions.

The procedure produces a row for each user of a version. Run **sp_iqversionuse** first on the coordinator to determine which versions should be released and the amount of space in KB to be released when the version is no longer in use. Connection IDs are displayed in the IQConn column for users connected to the coordinator. Version usage due to secondary servers is displayed as the secondary server name with connection ID 0.

The amount of space is expressed as a range because the actual amount typically depends on which other versions are released. The actual amount of space released can be anywhere between the values of MinKBRelease and MaxKBRelease. The oldest version always has MinKBRelease equal to MaxKBRelease.

The WasReported column is used in a multiplex setting. WasReported indicates whether version usage information has been sent from the secondary server to the coordinator.

WasReported is 0 initially on a coordinator for new versions. WasReported changes to 1 once the database server replicates version usage information back to the coordinator.

Run **sp_iqversionuse** on multiplex secondary servers to determine individual connections to secondary servers. Users from other servers are not displayed on a secondary server.

Table 142. sp_iqversionuse Columns

Column Name	Description
VersionID	In simplex databases, the VersionID is displayed as zero. For the multiplex coordinator, the VersionID is the same as the TxnID of the active transaction and VersionID is the same as the CmtID of a committed transaction. In multiplex secondary servers, the VersionID is the CmtID of the transaction that created the database version on the multiplex coordinator. It is used internally by the SAP Sybase IQ in-memory catalog and the SAP Sybase IQ transaction manager to uniquely identify a database version to all nodes within a multiplex database.
Server	The server to which users of this version are connected
IQConnID	The connection ID using this version
WasReported	Indicates whether the server has received usage information for this version
MinKBRelease	The minimum amount of space returned once this version is no longer in use
MaxKBRelease	The maximum amount of space returned once this version is no longer in use

Examples

Sample output from the **sp_iqversionuse** system procedure:

```
VersionID Server       IQConnID WasReported
========= ======       ======== ===========
        0 ab2ab_iqdemo        9           0

MinKBRelease  MaxKBRelease
============  ============
           0             0
```

The following examples show multiplex output. The oldest version 42648 is in use by connection 108 on the coordinator (*mpxw*). Committing or rolling back the transaction on connection 108 releases 7.9MB of space. Version 42686 is in use by secondary server (*mpxq*) according to output from the coordinator. Using the secondary server output, the actual connection is connection 31. The actual amount of space returned from releasing version 42686 depends on whether 42648 is released first.

WasReported is 0 for versions 42715 and 42728 on the coordinator because these are new versions that have not yet been replicated. Since version 42728 does not appear on the secondary server output, it has not yet been used by the secondary server.

Output returned when **sp_iqversionuse** executes on the coordinator *mpxw*:

```
call dbo.sp_iqversionuse
```

VersionID	Server	IQConn	WasReported	MinKBRelease	MaxKBRelease
42648	'mpxw'	108	1	7920	7920
42686	'mpxq'	0	1	7920	304
42702	'mpxq'	0	1	0	688
42715	‘mpxq'	0	0	0	688
42728	'mpxq'	0	0	0	688

Output returned when **sp_iqversionuse** executes on the secondary server (*mpxq*):

```
call dbo.sp_iqversionuse
```

VersionID	Server	IQConn	WasReported	MinKBRelease	MaxKBRelease
42686	'mpxq'	31	1	0	0
42715	'mpxq'	00	1	0	0

See also

- *sp_iqstatus Procedure* on page 566
- *sp_iqtransaction Procedure* on page 581

sp_iqview Procedure

Displays information about views in a database.

Syntax1

```
sp_iqview ([view_name],[view_owner],[view_type])
```

Syntax2

```
sp_iqview [view_name='viewname'],
[view_owner='viewowner' ],[view_type='viewtype' ]
```

Privileges

No specific system privileges are required to run this procedure.

Usage: Syntax1
`sp_iqview NULL,NULL,SYSTEM`

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example: `sp_iqview NULL,NULL,SYSTEM` and `sp_iqview deptview,NULL,'ALL'.`

Note: The *view_type* value ALL must be enclosed in single quotes in Syntax1.

Usage: Syntax2
The parameters can be specified in any order. Enclose them in single quotes.

The allowed values for the *view_type* parameter:

Table 143. sp_iqview view_type values

view_type value	Information displayed
SYSTEM	System views
ALL	User and system views
any other value	User views

Description
Specifying one of the parameters returns only the views with the specified view name or views that are owned by the specified user. Specifying more than one parameter filters the results by all of the parameters specified. Specifying no parameters returns all user views in a database.

Table 144. sp_iqview columns

Column name	Description
view_name	The name of the view
view_owner	The owner of the view
view_def	The view definition as specified in the **CREATE VIEW** statement
remarks	User comments added with the **COMMENT** statement

sp_iqview returns a view definition greater than 32K characters without truncation.

See also
- *sp_iqcolumn Procedure* on page 447
- *sp_iqconstraint Procedure* on page 455
- *sp_iqdatatype Procedure* on page 463
- *sp_iqevent Procedure* on page 484

- *sp_iqhelp Procedure* on page 489
- *sp_iqindex and sp_iqindex_alt Procedures* on page 496
- *sp_iqpkeys Procedure* on page 536
- *sp_iqprocparm Procedure* on page 540
- *sp_iq_reset_identity Procedure* on page 549
- *sp_iqtable Procedure* on page 575

sp_iqview Procedure Examples

Use the examples as a reference for **sp_iqview** usage.

The following variations in syntax both return information about the view `deptview`:

```
call sp_iqview('ViewSalesOrders')
```

```
sp_iqview view_name='ViewSalesOrders'
```

The following variations in syntax both return all views that are owned by view owner `GROUPO`:

```
sp_iqview NULL,GROUPO
```

```
sp_iqview view_owner='GROUPO'
```

view_name	view_owner	view_def	remarks
ViewSalesOrders	GROUPO	Create views GROUPO , ViewSalesOrders(ID, LineID, ProductID, Quantity, OrderDate, ShipDate, Region, SalesRepresentativeName	(NULL)

sp_iqwho Procedure

Displays information about all current users and connections, or about a particular user or connection.

Syntax

sp_iqwho `[ { `*`connhandle`*` | `*`user-name`*` } [, `*`arg-type`*` ] ]`

Privileges

Requires the DROP CONNECTION, MONITOR, and SERVER OPERATOR system privileges. Users without these system privileges must be granted `EXECUTE` permission to run the stored procedure.

Description

The **sp_iqwho** stored procedure displays information about all current users and connections, or about a particular user or connection.

Column name	Description
ConnHandle	The SA connection handle
IQConnID	The SAP Sybase IQ specific connection ID
Userid	The name of the user that opened the connection “ConnHandle”
BlockedOn	The connection on which a particular connection is blocked; 0 if not blocked on any connection
BlockUserid	The owner of the blocking connection; NULL if there is no blocking connection
ReqType	The type of the request made through the connection; DO_NOTHING if no command is issued
IQCmdType	The type of SAP Sybase IQ command issued from the connection; NONE if no command is issued
IQIdle	The time in seconds since the last SAP Sybase IQ command was issued through the connection; in case of no last SAP Sybase IQ command, the time since ‘01-01-2000’ is displayed
SAIdle	The time in seconds since the last SA request was issued through the connection; in case of no last SA command, the time since ‘01-01-2000’ is displayed
IQCursors	The number of active cursors in the connection; 0 if no cursors
IQThreads	The number of threads with the connection. At least one thread is started as soon as the connection is opened, so the minimum value for IQThreads is 1.
TempTableSpaceKB	The size of temporary table space in kilobytes; 0 if no temporary table space is used
TempWorkSpaceKB	The size of temporary workspace in kilobytes; 0 if no temporary workspace is used

Table 145. Mapping of sp_who and sp_iqwho columns

sp_who column	sp_iqwho column
fid	Family to which a lock belongs; omitted, as not applicable to SAP Sybase IQ
spid	ConnHandle, IQConnID
status	IQIdle, SAIdle

sp_who column	sp_iqwho column
loginame	Userid
origname	User alias; omitted, as not applicable to SAP Sybase IQ
hostname	Name of the host on which the server is running; currently not supported
blk_spid	BlockedOn
dbname	Omitted, as there is one server and one database for SAP Sybase IQ and they are the same for every connection
cmd	ReqType, IQCmdType
block_xloid	BlockUserid

Usage

Parameter	Description
connhandle	An integer representing the connection ID. If this parameter is specified, **sp_iqwho** returns information only about the specified connection. If the specified connection is not open, no rows are displayed in the output.
user-name	A char(255) parameter representing a user login name. If this parameter is specified, **sp_iqwho** returns information only about the specified user. If the specified user has not opened any connections, no rows are displayed in the output. If the specified user name does not exist in the database, **sp_iqwho** returns the error message `"User` *`user-name`* `does not exist"`
arg-type	The *arg-type* parameter is optional and can be specified only when the first parameter has been specified. The only value for *arg-type* is **"user"**. If the *arg-type* value is specified as **"user"**, **sp_iqwho** interprets the first parameter as a user name, even if the first parameter is numeric. If any value other than **"user"** is specified for *arg-type*, **sp_iqwho** returns the error `"Invalid parameter."` Enclose the *arg-type* value in double quotes.

If no parameters are specified, **sp_iqwho** displays information about all currently active connections and users.

Either a connection handle or a user name can be specified as the first **sp_iqwho** parameter. The parameters *connhandle* and *user-name* are exclusive and optional. Only one of these parameters can be specified at a time. By default, if the first parameter is numeric, the parameter is assumed to be a connection handle. If the first parameter is not numeric, it is assumed to be a user name.

SAP Sybase IQ allows numeric user names. The *arg-type* parameter directs **sp_iqwho** to interpret a numeric value in the first parameter as a user name. For example:

```
sp_iqwho 1, “user”
```

When the *arg-type* **“user”** is specified, **sp_iqwho** interprets the first parameter 1as a user name, not as a connection ID. If a user named `1` exists in the database, **sp_iqwho** displays information about connections opened by user `1`.

Syntax	Output
sp_iqwho	Displays all active connections
sp_iqwho 3	Displays information about connection 3
sp_iqwho “DBA”	Displays connections opened by user `DBA`
sp_iqwho 3, “user”	Interprets 3 as a user name and displays connections opened by user 3. If user 3 does not exist, returns the error “`User 3 does not exist`”
sp_iqwho non-existing-user	Returns error “`User non-existing-user does not exist`”
sp_iqwho 3, “xyz”	Returns the error “`Invalid parameter: xyz`”

sp_iqwho Procedure Example

Use the example as a reference for **sp_iqwho** usage.

Display all active connections:

```
ConnHandle IQConnID Userid ReqType      IQCmdType            Blocked
On BlockUserid IQCursors

12         118      DBA    CURSOR_OPEN  IQUTILITYOPENCURSOR  0
    (NULL)      0
13         119      shweta DO_NOTHING   NONE                 0
    (NULL)      0


IQThreads IQIdle   SAIdle TempTableSpaceKB TempWorkSpaceKB
1         1          0      0              0
1         16238757   470    0              0
```

sp_iqwho compatibility with Adaptive Server Enterprise

The SAP Sybase IQ **sp_iqwho** stored procedure incorporates the SAP Sybase IQ equivalents of columns displayed by the Adaptive Server Enterprise **sp_who** procedure.

Some Adaptive Server Enterprise columns are omitted, as they are not applicable to SAP Sybase IQ.

sp_iqworkmon Procedure

Controls collection of workload monitor usage information, and reports monitoring collection status. **sp_iqworkmon** collects information for all SQL statements.

Syntax

```
sp_iqworkmon [ 'action' ] [ , 'mode' ]
```

```
action = 'start' , 'stop' , 'status' , 'reset'
```

```
mode = 'index' , 'table' , 'column' , 'all'
```

For example:

```
sp_iqworkmon  'start' , 'all'
```

If one argument is specified, it can only be *action*. For example:

```
sp_iqworkmon  'stop'
```

Privileges

Requires the MONITOR system privilege. Users without the MONITOR system privilege must be granted `EXECUTE` permission to run the stored procedure.

Usage

Parameter	Description
action	Specifies the control action to apply. A value of *start* starts monitoring for the specified mode immediately. A value of *stop* stops monitoring immediately. A value of *status* (the default) displays the current status without changing state. The statistics are persisted until they are cleared with the *reset* argument, or until the server is restarted. Statistics collection does not automatically resume after a server restart, and it needs to be restarted using *start*.

Parameter	Description
mode	Specifies the type of monitoring to control. The INDEX, TABLE, and COLUMN keywords individually control monitoring of index usage, table usage, and column usage respectively. The default ALL keyword controls monitoring of all usage monitoring features simultaneously.

There is always a result set when you execute **sp_iqworkmon**. If you specify a specific mode (such as index), only the row for that mode appears.

Column name	Description
MonMode	Table, index, or column
Status	Started or stopped
Rowcount	Current number of rows collected

Example
Sample output from the **sp_iqworkmon** procedure:

```
MonMode     Status      Rowcount index       started     15
table       started     10 column      started     31
```

See also

- *sp_iqcolumnuse Procedure* on page 450
- *sp_iqindexadvice Procedure* on page 499
- *sp_iqindexuse Procedure* on page 512
- *sp_iqtableuse Procedure* on page 580
- *sp_iqunusedcolumn Procedure* on page 585
- *sp_iqunusedindex Procedure* on page 586
- *sp_iqunusedtable Procedure* on page 587

Alphabetical List of Catalog Stored Procedures

Catalog store stored procedures return result sets displaying database server, database, and connection properties in tabular form.

These procedures are owned by the `dbo` user ID. The PUBLIC role has EXECUTE permission on them.

sa_ansi_standard_packages system procedure

Returns information about the non-core SQL extensions used in a SQL statement.

Syntax

```
sa_ansi_standard_packages(
standard
, statement
)
```

Arguments

- ***standard* –** Use this LONG VARCHAR parameter to specify the standard to use for the core extensions. One of SQL:1999 or SQL:2003.
- ***statement* –** Use this LONG VARCHAR parameter to specify the SQL statement to evaluate.

Result set

Column name	Data type	Description
package_id	VARCHAR(10)	The feature identifier.
package_name	LONG VARCHAR	The feature name.

Remarks

If there are no non-core extensions used for the statement, the result set is empty.

Privileges

None

Side effects

None

sa_audit_string system procedure

Adds a string to the transaction log.

Syntax

```
sa_audit_string( string )
```

Arguments

- ***string* –** The VARCHAR(128) string of characters to add to the transaction log.

Remarks
If auditing is turned on, this system procedure adds a comment to the auditing information stored in the transaction log. The string can be a maximum of 128 characters.

Privileges
You must have the MANAGE AUDITING system privilege.

Side effects
None

Example

The following example uses sa_audit_string to add a comment to the transaction log:

```
CALL sa_audit_string( 'Auditing test' );
```

sa_char_terms System Procedure

Breaks a `CHAR` string into terms and returns each term as a row along with its position.

Syntax

```
sa_char_terms( 'char-string' [, 'text-config-name'
[, 'owner' ] ] ]     )
```

Parameters
char-string – the `CHAR` string you are parsing.

text-config-name – the text configuration object to apply when processing the string. The default value is 'default_char'.

owner – the owner of the specified text configuration object. The default value is DBA.

Description
You can use **sa_char_terms** to find out how a string is interpreted when the settings for a text configuration object are applied. This can be helpful when you want to know what terms would be dropped during indexing or from a query string.

Permissions
None.

Example
Return the terms in the `CHAR` string 'the quick brown fox jumped over the fence':

```
CALL sa_char_terms
( 'the quick brown fox jumped over the fence' );
```

Table 146. CHAR string interpretation

Term	Position
the	1
quick	2
brown	3
fox	4
jumped	5
over	6
the	7
fence	8

sa_checkpoint_execute system procedure

Allows the execution of shell commands during a checkpoint.

Syntax

```
sa_checkpoint_execute 'shell_commands'
```

Parameters

Parameter	Description
shell_commands	One or more user commands to be executed in a system shell. The shell commands are specific to the system shell. Commands are separated by a semicolon (;).

Privileges

Requires the CHECKPOINT or MANAGE ANY MIRROR SERVER system privilege to run the procedure.

Description

Allows users to execute shell commands to copy a running database from the middle of a checkpoint operation, when the server is quiescent. The copied database can be started and goes through normal recovery, similar to recovery following a system failure.

sa_checkpoint_execute initiates a checkpoint, and then executes a system shell from the middle of the checkpoint, passing the user commands to the shell. The server then waits for the shell to complete, creating an arbitrary size time window during which to copy database files. Most database activity stops while the checkpoint is executing, so the duration of the shell commands should be limited to acceptable user response time.

If the shell commands return a nonzero status, **sa_checkpoint_execute** returns an error.

Do not use the **sa_checkpoint_execute** with interactive commands, as the server must wait until the interactive command is killed. Supply override flags to disable prompting for any shell commands that might become interactive; in other words, the **COPY**, **MOVE**, and **DELETE** commands might prompt for confirmation.

The intended use of **sa_checkpoint_execute** is with disk mirroring, to split mirrored devices.

When using `sa_checkpoint_execute` to copy `iqdemo.*` files to another directory, all files are copied except the `.db` and `.log` files. Error -910 is returned.

This error not a product defect but a Windows limitation; the Windows copy command cannot copy catalog files while they are open by the database.

Example

Assuming you have created a subdirectory named `backup`, the following statement issues a checkpoint, copies all of the **iqdemo** database files to the backup subdirectory, and completes the checkpoint:

```
sa_checkpoint_execute 'cp iqdemo.* backup/'
```

sa_conn_activity system procedure

Returns the most recently-prepared SQL statement for each connection to the indicated database on the server.

Syntax

```
sa_conn_activity( [ connidparm ] )
```

Arguments

- ***connidparm* –** Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.

Result set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.
Name	VARCHAR(255)	Returns the name of the current connection. Temporary connection names have `INT:` prepended to the connection name

Column name	Data type	Description
Userid	VARCHAR(255)	Returns the user ID for the connection.
DBNumber	INTEGER	Returns the ID number of the database.
LastReqTime	VARCHAR(255)	Returns the time at which the last request for the specified connection started. This property can return an empty string for internal connections, such as events.
LastStatement	LONG VARCHAR	Returns the most recently prepared SQL statement for the current connection.

Remarks

If *connidparm* is less than zero, then information for the current connection is returned. If *connidparm* is not supplied or is NULL, then information is returned for all connections to all databases running on the database server.

The sa_conn_activity system procedure returns a result set consisting of the most recently-prepared SQL statement for the connection. Recording of statements must be enabled for the database server before calling sa_conn_activity. To do this, specify the -zl option when starting the database server, or execute the following:

```
CALL sa_server_option('RememberLastStatement','ON');
```

This procedure is useful when the database server is busy and you want to obtain information about the last SQL statement prepared for each connection. This feature can be used as an alternative to request logging.

Privileges

No privileges are required to execute this system procedure for the current connection ID. To execute this system procedure for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Side effects

None

Example

The following example uses the sa_conn_activity system procedure to display the most recently-prepared SQL statement for each connection.

```
CALL sa_conn_activity( );
```

Number	Name	Userid	DBNumber	...
1,949	SQL_DBC_117acc40	DBA	0	...
1,948	setup	User1	0	...
...	...	...	...	...

sa_conn_info system procedure

Reports connection property information.

Syntax

```
sa_conn_info( [ connidparm ] )
```

Arguments

- ***connidparm* –** This optional INTEGER parameter specifies the connection ID number. The default is NULL.

Result set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.
Name	VARCHAR(255)	Returns the connection ID (a number) for the current connection. Temporary connection names have `INT:` prepended to the connection name.
Userid	VARCHAR(255)	Returns the user ID for the connection.
DBNumber	INTEGER	Returns the ID number of the database.

Column name	Data type	Description
LastReqTime	VARCHAR(255)	Returns the time at which the last request for the specified connection started. This property can return an empty string for internal connections, such as events.
ReqType	VARCHAR(255)	Returns the type of the last request. If a connection has been cached by connection pooling, its ReqType value is CONNECT_POOL_CACHE.
CommLink	VARCHAR(255)	Returns the communication link for the connection. This is one of the network protocols supported by SAP Sybase IQ, or local for a same-computer connection.
NodeAddr	VARCHAR(255)	Returns the address of the client in a client/server connection.
ClientPort	INTEGER	Returns the client's TCP/IP port number or 0 if the connection isn't a TCP/IP connection.
ServerPort	INTEGER	Returns the database server's TCP/IP port number or 0.
BlockedOn	INTEGER	Returns zero if the current connection isn't blocked, or if it is blocked, the connection number on which the connection is blocked because of a locking conflict.

Column name	Data type	Description
LockRowID	UNSIGNED BIGINT	Returns the identifier of the locked row. LockRowID is NULL if the connection is not waiting on a lock associated with a row (that is, it is not waiting on a lock, or it is waiting on a lock that has no associated row).
LockIndexID	INTEGER	Returns the identifier of the locked index. LockIndexID is -1 if the lock is associated with all indexes on the table in LockTable. LockIndexID is NULL if the connection is not waiting on a lock associated with an index (that is, it is not waiting on a lock, or it is waiting on a lock that has no associated index).
LockTable	VARCHAR(255)	Returns the name of the table associated with a lock if the connection is currently waiting for a lock. Otherwise, LockTable returns an empty string.
UncommitOps	INTEGER	Returns the number of uncommitted operations.
ParentConnection	INTEGER	Returns the connection ID of the connection that created a temporary connection to perform a database operation (such as performing a backup or creating a database). For other types of connections, this property returns NULL.

Remarks

If *connidparm* is less than zero, then a result set consisting of connection properties for the current connection is returned. If *connidparm* is not supplied or is NULL, then connection properties are returned for all connections to all databases running on the database server.

In a block situation, the BlockedOn value returned by this procedure allows you to check which users are blocked, and who they are blocked on. The sa_locks system procedure can be used to display the locks held by the blocking connection.

For more information based on any of these properties, you can execute something similar to the following:

```
SELECT *, DB_NAME( DBNumber ),
   CONNECTION_PROPERTY( 'LastStatement', Number )
   FROM sa_conn_info( );
```

The value of LockRowID can be used to look up a lock in the output of the sa_locks procedure.

The value in LockIndexID can be used to look up a lock in the output of the sa_locks procedure. Also, the value in LockIndexID corresponds to the primary key of the ISYSIDX system table, which can be viewed using the SYSIDX system view.

Every lock has an associated table, so the value of LockTable can be used to unambiguously determine whether a connection is waiting on a lock.

Privileges

No privileges are required to execute this system procedure for the current connection ID. To execute this system procedure for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Side effects

None

Examples

The following example uses the sa_conn_info system procedure to return a result set summarizing connection properties for all connections to the server.

```
CALL sa_conn_info( );
```

Number	Name	Userid	DBNumber	...
79	SQL_DBC_10dcf810	DBA	0	...
46	setup	User1	0	...
...	...	...	...	...

The following example uses the sa_conn_info system procedure to return a result set showing which connection created a temporary connection.

```
SELECT Number, Name, ParentConnection FROM sa_conn_info();
```

Connection 8 created the temporary connection that executed a CREATE DATABASE statement.

```
Number      Name                    ParentConnection
------------------------------------------------
```

```
1000000048  INT: CreateDB      8
9           SQL_DBC_14675af8   (NULL)
8           SQL_DBA_152d5ac0   (NULL)
```

sa_conn_list system procedure

Returns a result set containing connection IDs.

Syntax

```
sa_conn_list ([ connidparm ] [ ,dbidparm])
```

Arguments

- *connidparm* Use this optional INTEGER parameter to specify the connection ID number.
- *dbidparm* Use this optional INTEGER parameter to specify the database ID number.

Result Set

Column Name	Data Type	Description
Number	INTEGER	The connection ID number.

sa_conn_properties system procedure

Reports connection property information.

Syntax

```
sa_conn_properties( [ connidparm ] )
```

Arguments

- ***connidparm* –** Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.

Result set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.
PropNum	INTEGER	Returns the connection property number.
PropName	VARCHAR(255)	Returns the connection property name.

Column name	Data type	Description
PropDescription	VARCHAR(255)	Returns the connection property description.
Value	LONG VARCHAR	Returns the connection property value.

Remarks
Returns the connection ID as Number, and the PropNum, PropName, PropDescription, and Value for each available connection property. Values are returned for all connection properties, database option settings related to connections, and statistics related to connections. Valid properties with NULL values are also returned.

If *connidparm* is less than zero, then property values for the current connection are returned. If *connidparm* is not supplied or is NULL, then property values are returned for all connections to the current database.

Privileges
No privileges are required to execute this system procedure for the current connection ID. To execute this system procedure for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Side effects
None

Examples

The following example uses the sa_conn_properties system procedure to return a result set summarizing connection property information for all connections.

```
CALL sa_conn_properties( );
```

Number	PropNum	PropName	...
79	37	ClientStmtCacheHits	...
79	38	ClientStmtCacheMiss-es	...
...	...	...	...

This example uses the sa_conn_properties system procedure to return a list of all connections, in decreasing order by CPU time*:

```
SELECT Number AS connection_number,
    CONNECTION_PROPERTY ( 'Name', Number ) AS connection_name,
    CONNECTION_PROPERTY ( 'Userid', Number ) AS user_id,
  CAST ( Value AS NUMERIC ( 30, 2 ) ) AS approx_cpu_time
  FROM sa_conn_properties( )
  WHERE PropName = 'ApproximateCPUTime'
  ORDER BY approx_cpu_time DESC;
```

*Example courtesy of Breck Carter, RisingRoad Professional Services (*http://www.risingroad.com*).

sa_db_info system procedure

Reports database property information.

Syntax

```
sa_db_info( [ dbidparm ] )
```

Arguments

- ***dbidparm*** **–** Use this optional INTEGER parameter to specify the database ID number. The default is NULL.

Result set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.
Alias	VARCHAR(255)	Returns the database name.
File	VARCHAR(255)	Returns the file name of the database root file, including path.
ConnCount	INTEGER	Returns the number of connections to the database. The property value does not include connections used for internal operations, but it does include connections used for events and external environment support.
PageSize	INTEGER	Returns the page size of the database, in bytes.
LogName	VARCHAR(255)	Returns the file name of the transaction log, including path.

Remarks

If you specify a database ID, sa_db_info returns a single row containing the Number, Alias, File, ConnCount, PageSize, and LogName for the specified database.

If *dbidparm* is greater than zero, then properties for the supplied database are returned. If *dbidparm* is less than zero, then properties for the current database are returned. If *dbidparm* is not supplied or is NULL, then properties for all databases running on the database server are returned.

Privileges

No privileges are required to execute this system procedure for the current database. To execute this system procedure for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

Side effects

None

Example

The following statement returns a row for each database that is running on the server:

```
CALL sa_db_info( );
```

Property	Value
Number	0
Alias	iqdemo
File	`C:\ProgramData\SybaseIQ\demo\iqdemo.db`
ConnCount	3
PageSize	4096
LogName	`C:\ProgramData\SybaseIQ\demo\iqdemo.log`

sa_db_properties system procedure

Reports database property information.

Syntax

```
sa_db_properties( [ dbidparm ] )
```

Arguments

- ***dbidparm*** **–** Use this optional INTEGER parameter to specify the database ID number. The default is NULL.

Result set

Column name	Data type	Description
Number	INTEGER	The database ID number.
PropNum	INTEGER	The database property number.
PropName	VARCHAR(255)	The database property name.
PropDescription	VARCHAR(255)	The database property description.
Value	LONG VARCHAR	The database property value.

Remarks

If you specify a database ID, the sa_db_properties system procedure returns the database ID number and the PropNum, PropName, PropDescription, and Value for each available database property. Values are returned for all database properties and statistics related to databases. Valid properties with NULL values are also returned.

If *dbidparm* is greater than zero, then database properties for the supplied database are returned. If *dbidparm* is less than zero, then database properties for the current database are returned. If *dbidparm* is not supplied or is NULL, then database properties for all databases running on the database server are returned.

Privileges

No privileges are required to execute this system procedure for the current database. To execute this system procedure for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

Side effects

None

Example

The following example uses the sa_db_properties system procedure to return a result set summarizing database properties for all databases when the invoker has SERVER OPERATOR or MONITOR system privilege. Otherwise, database properties for the current database are returned.

```
CALL sa_db_properties( );
```

Number	PropNum	PropName	...
0	0	ConnCount	...
0	1	IdleCheck	...

Number	PropNum	PropName	...
0	2	IdleWrite	...
...	...	...	...

The following example uses the sa_db_properties system procedure to return a result set summarizing database properties for a second database.

```
CALL sa_db_properties( 1 );
```

sa_describe_shapefile System Procedure

Describes the names and types of columns contained in an ESRI shapefile. This system feature is for use with the spatial data features.

Syntax

```
sa_describe_shapefile (
    shp_filename
    , srid
[, encoding ]
)
```

Arguments

- **shp_filename –** A VARCHAR(512) parameter that identifies the location of the ESRI shapefile. The file name must have the extension .shp and must have an associated .dbf file with the same base name located in the same directory. The path is relative to the database server, not the client application.
- **srid –** An INTEGER parameter that identifies the SRID for the geometries in the shapefile. Specify NULL to indicate the column can store multiple SRIDs. Specifying NULL limits the operations that can be performed on the geometry values.
- **encoding –** An optional VARCHAR(50) parameter that identifies the encoding to use when reading the shapefile. The default is NULL. When encoding is NULL, the ISO-8859-1 character set is used.

Result Set

Column Name	Data Type	Description
column_number	INTEGER	The ordinal position of the column described by this row, starting at 1.
name	VARCHAR(128)	The name of the column.

Column Name	Data Type	Description
domain_name_with_size	VARCHAR(160)	The data type name, including size and precision (as used in CREATE TABLE or CAST functions).

Remarks

The sa_describe_shapefile system procedure is used to describe the name and type of columns in an ESRI shapefile. This information can be used to create a table to load data from a shapefile using the LOAD TABLE or INPUT statements. Alternately, this system procedure can be used to read a shapefile by specifying the WITH clause for OPENSTRING...FORMAT SHAPEFILE.

Privileges

- If the -gl database option is set to DBA, you requires one of the following:
  - ALTER ANY TABLE system privilege
  - ALTER ANY OBJECT system privilege
  - LOAD ANY TABLE system privilege
  - READ FILE system privilege
- If the -gl database option is set to ALL, no privileges are required.
- If the -gl database option is set to NONE, you must have the READ FILE system privilege.

Example

The following example displays a string that was used to create a table for storing shapefile data:

```
BEGIN
     DECLARE create_cmd LONG VARCHAR;
    SELECT 'create table if not exists esri_load( record_number int
primary key, ' ||
         (SELECT list( name || ' ' || domain_name_with_size, ', '
ORDER BY column_number )
     FROM sa_describe_shapefile( 'c:\\esri\\tgr36069trt00.shp',
1000004326 )
     WHERE column_number > 1 ) || ' )'
     INTO create_cmd;
     SELECT create_cmd;
     EXECUTE IMMEDIATE create_cmd;
END
```

You can load the shapefile data into the table using the following statement (provided that you have the LOAD ANY TABLE system privilege and that the -gl database option has not been set to NONE):

```
LOAD TABLE esri_load
USING FILE 'c:\\esri\\tgr36069trt00.shp'
FORMAT SHAPEFILE;
```

sa_dependent_views system procedure

Returns the list of all dependent views for a given table or view.

Syntax

```
sa_dependent_views(
[ tbl_name
[, owner_name ] ]
)
```

Arguments

- ***tbl_name* –** Use this optional CHAR(128) parameter to specify the name of the table or view. The default is NULL.
- ***owner_name* –** Use this optional CHAR(128) parameter to specify the owner for *tbl_name*. The default is NULL.

Result set

Column name	Data type	Description
table_id	UNSIGNED INTEGER	The object ID of the table or view.
dep_view_id	UNSIGNED INTEGER	The object ID of the dependent views.

Remarks

Use this procedure to obtain the list of IDs of tables and their dependent views.

No errors are generated if no existing tables satisfy the specified criteria for table and owner names. The following conditions also apply:

- If both *owner* and *tbl_name* are NULL, information is returned on all tables that have dependent views.
- If *tbl_name* is NULL but *owner* is specified, information is returned on all tables owned by the specified owner.
- If *tbl_name* is specified but *owner* is NULL, information is returned on any one of the tables with the specified name.

Privileges

None

Side effects

None

Examples

In this example, the sa_dependent_views system procedure is used to obtain the list of IDs for the views that are dependent on the SalesOrders table. The procedure returns the table_id for SalesOrders, and the dep_view_id for the dependent view, ViewSalesOrders.

```
CALL sa_dependent_views( 'SalesOrders' );
```

In this example, the sa_dependent_views system procedure is used in a SELECT statement to obtain the list of names of views dependent on the SalesOrders table. The procedure returns the ViewSalesOrders view.

```
SELECT t.table_name FROM SYSTAB t,
sa_dependent_views( 'SalesOrders' ) v
WHERE t.table_id = v.dep_view_id;
```

sa_disable_auditing_type system procedure

Disables auditing of specific events.

Syntax

```
sa_disable_auditing_type( types )
```

Arguments

- ***types*** **–** Use this VARCHAR(128) parameter to specify a comma-delimited string containing one or more of the following values:
  - **all –** disables all types of auditing.
  - **connect –** disables auditing of both successful and failed connection attempts.
  - **connectFailed –** disables auditing of failed connection attempts.
  - **DDL –** disables auditing of DDL statements.
  - **options –** disables auditing of public options.
  - **permission –** disables auditing of permission checks, user checks, and SETUSER statements.
  - **permissionDenied –** disables auditing of failed permission and user checks.
  - **triggers –** disables auditing in response to trigger events.

Remarks

You can use the sa_disable_auditing_type system procedure to disable auditing of one or more categories of information.

Setting this option to all disables all auditing. You can also disable auditing by setting the PUBLIC.auditing option to Off.

Privileges

You must have the SET ANY SECURITY OPTION system privilege.

Side effects
None

Example

To disable all auditing:

```
CALL sa_disable_auditing_type( 'all' );
```

sa_disk_free_space system procedure

Reports information about space available for a transaction log, transaction log mirror, and/or temporary file.

Syntax

```
sa_disk_free_space( [ p_dbspace_name ] )
```

Arguments

- ***p_dbspace_name* –** Use this VARCHAR(128) parameter to specify the name of a transaction log file, transaction log mirror file, or temporary file. The default is NULL.

  Specify SYSTEM to get information about the main database file, TEMPORARY or TEMP to get information about the temporary file, TRANSLOG to get information about the transaction log, or TRANSLOGMIRROR to get information about the transaction log mirror.

Result set

Column name	Data type	Description
dbspace_name	VARCHAR(128)	This is the transaction log file, transaction log mirror file, or temporary file.
free_space	UNSIGNED BIGINT	The number of free bytes on the volume.
total_space	UNSIGNED BIGINT	The total amount of disk space available on the drive.

Remarks
If the *p_dbspace_name* parameter is not specified or is NULL, then the result set contains one row for each of the transaction log, transaction log mirror, and temporary file, if they exist. If *p_dbspace_name* is specified, then exactly one or zero rows are returned (zero if log or mirror is specified and there is no log or mirror file).

Privileges

You must have the MANAGE ANY DBSPACE system privilege.

Side effects

None

Example

The following example uses the sa_disk_free_space system procedure to return a result set containing information about available space.

```
CALL sa_disk_free_space( );
```

dbspace_name	free_space	total_space
system	10952101888	21410402304
translog	10952101888	21410402304
temporary	10952101888	21410402304

sa_enable_auditing_type system procedure

Enables auditing and specifies which events to audit.

Syntax

```
sa_enable_auditing_type( types )
```

Arguments

- ***types*** **–** Use this VARCHAR(128) parameter to specify a comma-delimited string containing one or more of the following values:
  - **all –** enables all types of auditing.
  - **connect –** enables auditing of both successful and failed connection attempts.
  - **connectFailed –** enables auditing of failed connection attempts.
  - **DDL –** enables auditing of DDL statements.
  - **options –** enables auditing of public options.
  - **permission –** enables auditing of permission checks, user checks, and SETUSER statements.
  - **permissionDenied –** enables auditing of failed permission and user checks.
  - **triggers –** enables auditing after a trigger event.

Remarks

sa_enable_auditing_type works with the PUBLIC.auditing option to enable auditing of specific types of information.

If you set the PUBLIC.auditing option to On, and do not specify which type of information to audit, the default setting (all) takes effect. In this case, all types of auditing information are recorded.

If you set the PUBLIC.auditing option to On, and disable all types of auditing using sa_disable_auditing_type, no auditing information is recorded. To re-establish auditing, you must use sa_enable_auditing_type to specify which type of information you want to audit.

If you set the PUBLIC.auditing option to Off, then no auditing information is recorded, regardless of the sa_enable_auditing_type setting.

Privileges

You must have the SET ANY SECURITY OPTION system privilege.

Side effects

None

Example

To enable only option auditing:

```
CALL sa_enable_auditing_type( 'options' );
```

sa_eng_properties system procedure

Reports database server property information.

Syntax

```
sa_eng_properties( )
```

Result set

Column name	Data type	Description
PropNum	INTEGER	The database server property number.
PropName	VARCHAR(255)	The database server property name.
PropDescription	VARCHAR(255)	The database server property description.
Value	LONG VARCHAR	The database server property value.

Remarks
Returns the PropNum, PropName, PropDescription, and Value for each available server property. Values are returned for all database server properties and statistics related to database servers.

Privileges
None

Side effects
None

Example

The following statement returns a set of available server properties

```
CALL sa_eng_properties( );
```

PropNum	PropName	...
1	IdleWrite	...
2	IdleChkPt	...
...	...	...

sa_external_library_unload System Procedure

Unloads an external library.

Syntax

```
sa_external_library_unload ( [ 'external-library' ] )
```

Parameters
external-library – optionally use this `LONG VARCHAR` parameter to specify the name of a library to be unloaded. If no library is specified, all external libraries that are not in use are unloaded.

Description
If an external library is specified, but is in use or is not loaded, an error is returned. If no parameter is specified, an error is returned if no loaded external libraries are found.

Privileges
MANAGE ANY EXTERNAL OBJECT system privilege required.

Examples
Unload an external library called `myextlib.dll`:

```
CALL sa_external_library_unload( 'myextlib.dll' );
```

Unload all libraries that are not currently in use:

```
CALL sa_external_library_unload();
```

sa_flush_cache system procedure

Empties all pages for the current database in the database server cache.

Syntax

```
sa_flush_cache( )
```

Remarks

Database administrators can use this procedure to empty the contents of the database server cache for the current database. This is useful in performance measurement to ensure repeatable results.

Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

None

Example

The following example empties all pages for the current database in the database server cache.

```
CALL sa_flush_cache( );
```

sa_get_user_status system procedure

Allows you to determine the current status of users.

Syntax

```
sa_get_user_status( )
```

Result set

Column name	Data type	Description
user_id	UNSIGNED INTEGER	A unique number identifying the user.
user_name	CHAR(128)	The name of the user.
connections	INTEGER	The current number of connections by this user.

Column name	Data type	Description
failed_logins	UNSIGNED INTEGER	The number of failed login attempts made by the user.
last_login_time	TIMESTAMP	The local time that the user last logged in.
locked	TINYINT	Indicates if the user account is locked.
reason_locked	LONG VARCHAR	The reason the account is locked.
user_dn	CHAR(1024)	The Distinguished Name (DN) for a user ID connecting to an LDAP server.
user_dn_cached_at	TIMESTAMP	The local time that the DN was stored.
password_change_state	BIT	A value that indicates whether a dual password change is in progress (0=No, 1=Yes). The default is 0.
password_change_first_user	UNSIGNED INTEGER	The user_id of the user who set the first part of a dual password; otherwise NULL.
password_change_second_user	UNSIGNED INTEGER	The user_id of the user who set the second part of a dual password; otherwise NULL.
user_dn	CHAR(1024)	The distinguished name (DN) of the user.
user_dn_cached_at	TIMESTAMP	The date and time the distinguished name was found.

Remarks

This procedure returns a result set that shows the current status of users. In addition to basic user information, the procedure includes a column indicating if the user has been locked out and a column with a reason for the lockout. Users can be locked out for the following reasons: locked due to policy, password expiry, or too many failed attempts.

If the user is authenticated using LDAP User Authentication, the output includes the user's distinguished name and the date and time that the distinguished name was found.

Privileges
You can view information about yourself; no privilege is required. You must have the MANAGE ANY USER system privilege to view information about other users.

Side effects
None

Example

The following example uses the sa_get_user_status system procedure to return the status of database users.

```
CALL sa_get_user_status;
```

sa_get_ldapserver_status System Procedure

Determines the current status of the LDAP server configuration object.

Syntax
sa_get_ldapserver_status()

Arguments
None

Result Set

Column Name	Data Type	Description
ldsrv_id	UNSIGNED BIGINT	A unique identifier for the LDAP server configuration object that is the primary key and is used by the login policy to refer to the LDAP server.
ldsrv_name	CHAR(128)	The name assigned to the LDAP server configuration object.
ldsrv_state	CHAR(9)	Read-only state of the LDAP server: 1 – RESET 2 – READY 3 – ACTIVE 4 – FAILED 5 – SUSPENDED A numeric value is stored in system table; a corresponding text value appears in the system view.

Column Name	Data Type	Description
ldsrv_last_state_change	TIMESTAMP	Indicates the time the last state change occurred. The value is stored in Coordinated Universal Time (UTC), regardless of the local time zone of the LDAP server.

Remarks
To see SYSLDAPSERVER column values before a checkpoint occurs and the contents of memory are written to the catalog on disk. The updates to the catalog columns ldsrv_state and ldsrv_last_state_change occur asynchronously during checkpoint to the LDAP server object as the result of an event that changes the LDAP server object state, such as a failed connection due to a failed LDAP directory server. The LDAP server object state reflects the state of the LDAP directory server.

Privileges
None.

sa_list_external_library system procedure

Lists the external libraries currently loaded in the server.

Syntax

```
sa_list_external_library ( )
```

Description
Returns a list of external libraries loaded in the engine along with their reference count.

The reference count is the number of instances of the library in the engine. An external library can be unloaded by executing the procedure **sa_external_library_unload**, only if its reference count is 0.

Privileges
MANAGE ANY EXTERNAL OBJECT system privilege required.

Example
List the external libraries and their reference count:

```
CALL sa_list_external_library()
```

sa_locks System Procedure

Displays all locks in the database.

Syntax

```
sa_locks (
  [ connection
  [, creator
  [, table_name
  [, max_locks ] ] ] ]
)
```

Arguments

- **connection –** Use this INTEGER parameter to specify a connection ID number. The procedure returns lock information only about the specified connection. The default value is 0 (or NULL), in which case information is returned about all connections.
- **creator –** Use this CHAR(128) parameter to specify a user ID. The procedure returns information only about the tables owned by the specified user. The default value for the creator parameter is NULL. When this parameter is set to NULL, sa_locks returns the following information:
  - if the table_name parameter is unspecified, locking information is returned for all tables in the database
  - if the table_name parameter is specified, locking information is returned for tables with the specified name that were created by the current user
- **table_name –** Use this CHAR(128) parameter to specify a table name. The procedure returns information only about the specified tables. The default value is NULL, in which case information is returned about all tables.
- **max_locks –** Use this INTEGER parameter to specify the maximum number of locks for which to return information. The default value is 1000. The value -1 means return all lock information

Result Set

Column Name	Data Type	Description
conn_name	VARCHAR(128)	The name of the current connection.
conn_id	INTEGER	The connection ID number.
user_id	CHAR(128)	The user ID for the connection.
table_type	CHAR(6)	The type of table. This type is either BASE for a table, GLBTMP for global temporary table, or MVIEW for a materialized view.

Column Name	Data Type	Description
creator	VARCHAR(128)	The owner of the table.
table_name	VARCHAR(128)	The table on which the lock is held.
index_id	INTEGER	The index ID or NULL.
lock_class	CHAR(8)	The lock class. One of Schema, Row, Table, or Position.
lock_duration	CHAR(11)	The duration of the lock. One of Transaction, Position, or Connection.
lock_type	CHAR(9)	The lock type (this is dependent on the lock class).
row_identifier	UNSIGNED BIGINT	The identifier for the row. This is either an 8-byte row identifier or NULL.

Remarks

The sa_locks procedure returns a result set containing information about all the locks in the database. The value in the lock_type column depends on the lock classification in the lock_class column. The following values can be returned:

Lock Class	Lock Types	Comments
Schema	Shared (shared schema lock) Exclusive (exclusive schema lock)	For schema locks, the row_identifier and index ID values are NULL.

Lock Class	Lock Types	Comments
Row	Read (read lock) Intent (intent lock) ReadPK (read lock) Write (write lock) WriteNoPK (write lock) Surrogate (surrogate lock)	Row read locks can be short-term locks (scans at isolation level 1) or can be long-term locks at higher isolation levels. The lock_duration column indicates whether the read lock is of short duration because of cursor stability (Position) or long duration, held until COMMIT/ROLLBACK (Transaction). Row locks are always held on a specific row, whose 8-byte row identifier is reported as a 64-bit integer value in the row_identifier column. A surrogate lock is a special case of a row lock. Surrogate locks are held on surrogate entries, which are created when referential integrity checking is delayed. There is not a unique surrogate lock for every surrogate entry created in a table. Rather, a surrogate lock corresponds to the set of surrogate entries created for a given table by a given connection. The row_identifier value is unique for the table and connection associated with the surrogate lock. If required, key and non-key portions of a row can be locked independently. A connection can obtain a read lock on the key portion of a row for shared (read) access so that other connections can still obtain write locks on other non-key columns of a row. Updating non-key columns of a row does not interfere with the insertion and deletion of foreign rows referencing that row.
Table	Shared (shared table lock) Intent (intent to update table lock) Exclusive (exclusive table lock)	None
Position	Phantom (phantom lock) Insert (insert lock)	Usually a position lock is also held on a specific row, and that row's 64-bit row identifier appears in the row_identifier column in the result set. However, Position locks can be held on entire scans (index or sequential), in which case the row_identifier column is NULL.

A position lock can be associated with a sequential table scan, or an index scan. The index_id column indicates whether the position lock is associated with a sequential scan. If the position lock is held because of a sequential scan, the index_id column is NULL. If the position lock is held as the result of a specific index scan, the index identifier of that index is listed in the index_id column. The index identifier corresponds to the primary key of the ISYSIDX system table, which can be viewed using the SYSIDX view. If the position lock is held for scans over all indexes, the index ID value is -1.

Privileges

Requires the MONITOR system privilege.

Example

You can execute the following query to identify locks.

```
CALL sa_locks( );
```

sa_make_object system procedure

Ensures that a skeletal instance of an object exists before executing an ALTER statement.

Syntax

```
sa_make_object(
objtype
, objname
[, owner
[, tabname ]  ]
)
```

```
objtype:
'procedure'
| 'function'
| 'view'
| 'trigger'
| 'service'
| 'event'
```

Arguments

- ***objtype*** **–** Use this CHAR(30) parameter to specify the type of object being created. If objtype is `'trigger'`, this argument specifies the owner of the table on which the trigger is to be created.
- ***objname*** **–** Use this CHAR(128) parameter to specify the name of the object to be created.
- ***owner*** **–** Use this optional CHAR(128) parameter to specify the owner of the object to be created. The default is NULL.
- ***tabname*** **–** This CHAR(128) parameter is required only if objtype is `'trigger'`, in which case you use it to specify the name of the table on which the trigger is to be created. The default is NULL.

Remarks

This procedure is useful in scripts that are run repeatedly to create or modify a database schema. A common problem in such scripts is that the first time they are run, a CREATE statement must be executed, but subsequent times an ALTER statement must be executed. This procedure avoids the necessity of querying the system views to find out whether the object exists.

For procedures, functions, views, triggers, you can now use the OR REPLACE clause instead of this system procedure.

To use the procedure, follow it by an ALTER statement that contains the entire object definition.

Privileges

You must have the required privileges as follows:

- **Procedures or functions owned by the invoker –** CREATE PROCEDURE, CREATE ANY PROCEDURE, or CREATE ANY OBJECT system privilege
- **Procedures or functions owned by other users –** CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege
- **Services –** MANAGE ANY WEB SERVICE system privilege
- **Events –** MANAGE ANY EVENT or CREATE ANY OBJECT system privilege
- **Views owned by the invoker –** CREATE VIEW, CREATE ANY VIEW, or CREATE ANY OBJECT system privilege
- **Views owned by other users –** CREATE ANY VIEW or CREATE ANY OBJECT system privilege
- **Triggers –** If the trigger is on a table owned by you, you must have either the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege.

  If the trigger is on a table owned by another user, you must have either the CREATE ANY TRIGGER or the CREATE ANY OBJECT system privilege. Additionally, you must have one of the following:

  - ALTER ANY TABLE privilege
  - ALTER ANY OBJECT system privilege
  - ALTER permission on the table on which the trigger is being created.

Side effects

Automatic commit

Examples

The following statements ensure that a skeleton procedure definition is created, define the procedure, and grant privileges on it. A script file containing these instructions could be run repeatedly against a database without error.

```
CALL sa_make_object( 'procedure', 'myproc' );
ALTER PROCEDURE myproc( in p1 INT, in p2 CHAR(30) )
```

```
BEGIN
    // ...
END;
GRANT EXECUTE ON myproc TO public;
```

The following example uses the sa_make_object system procedure to add a skeleton web service.

```
CALL sa_make_object( 'service', 'my_web_service' );
```

sa_nchar_terms System Procedure

Breaks an `NCHAR` string into terms and returns each term as a row along with its position.

Syntax

```
sa_nchar_terms( 'char-string' [ , 'text-config-
name' [, 'owner' ] ] ]     )
```

Parameters

char-string – the `NCHAR` string you are parsing.

text-config-name – the text configuration object to apply when processing the string. The default value is 'default_nchar'.

owner – the owner of the specified text configuration object. The default value is DBA.

Description

You can use **sa_nchar_terms** to find out how a string is interpreted when the settings for a text configuration object are applied. This can be helpful when you want to know what terms would be dropped during indexing or from a query string.

The syntax for **sa_nchar_terms** is similar to the syntax for the **sa_char_terms** system procedure.

Note: The `NCHAR` data type is supported only for **IN SYSTEM** tables.

Permissions

None.

sa_performance_diagnostics System Procedure

Returns a summary of request timing information for all connections when the database server has request timing logging enabled.

Syntax

```
sa_performance_diagnostics( )
```

Result

Column Name	Data Type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.
Name	VARCHAR(255)	Returns the name of the current connection. You can specify a connection name using the ConnectionName (CON) connection parameter. The following names are used for temporary connections created by the database server: • INT:ApplyRecovery • INT:BackupDB • INT:Checkpoint • INT:Cleaner • INT:CloseDB • INT:CreateDB • INT:CreateMirror • INT:DelayedCommit • INT:DiagRcvr • INT:DropDB • INT:EncryptDB • INT:Exchange • INT:FlushMirrorLog • INT:FlushStats • INT:HTTPReq • INT:PromoteMirror • INT:PurgeSnapshot • INT:ReconnectMirror • INT:RecoverMirror • INT:RedoCheckpoint • INT:RefreshIndex • INT:ReloadTrigger • INT:RenameMirror • INT:RestoreDB • INT:StartDB • INT:VSS
Userid	VARCHAR(255)	Returns the user ID for the connection.

Column Name	Data Type	Description
DBNumber	INTEGER	Returns the ID number of the database.
LoginTime	TIMESTAMP	Returns the date and time the connection was established.
TransactionStart-Time	TIMESTAMP	Returns a string containing the time the database was first modified after a COMMIT or ROLLBACK, or an empty string if no modifications have been made to the database since the last COMMIT or ROLLBACK.
LastReqTime	TIMESTAMP	Returns the time at which the last request for the specified connection started. This property can return an empty string for internal connections, such as events.
ReqType	VARCHAR(255)	Returns the type of the last request. If a connection has been cached by connection pooling, its ReqType value is CONNECT_POOL_CACHE.
ReqStatus	VARCHAR(255)	Returns the status of the request. It can be one of the following values: • **Idle –** The connection is not currently processing a request. • **Unscheduled –** The connection has work to do and is waiting for an available database server worker. • **BlockedIO –** The connection is blocked waiting for an I/O. • **BlockedContention –** The connection is blocked waiting for access to shared database server data structures. • **BlockedLock –** The connection is blocked waiting for a locked object. • **Executing –** The connection is executing a request. The values marked with an asterisk (*) are only returned when logging of request timing information has been turned on for the database server using the -zt server option. If request timing information is not being logged (the default), the values are reported as Executing.

Column Name	Data Type	Description
ReqTimeUnscheduled	DOUBLE	Returns the amount of unscheduled time, or NULL if the -zt option was not specified.
ReqTimeActive	DOUBLE	Returns the amount of time, in seconds, spent processing requests, or NULL if the -zt option was not specified.
ReqTimeBlockIO	DOUBLE	Returns the amount of time, in seconds, spent waiting for I/O to complete, or NULL if the -zt option was not specified.
ReqTimeBlockLock	DOUBLE	Returns the amount of time, in seconds, spent waiting for a lock, or NULL if the -zt option was not specified.
ReqTimeBlockContention	DOUBLE	Returns the amount of time, in seconds, spent waiting for atomic access, or NULL if the RequestTiming server property is set to Off.
ReqCountUnscheduled	INTEGER	Returns the number of times the connection waited for scheduling, or NULL if the -zt option was not specified.
ReqCountActive	INTEGER	Returns the number of requests processed, or NULL if the RequestTiming server property is set to Off.
ReqCountBlockIO	INTEGER	Returns the number of times the connection waited for I/O to complete, or NULL if the -zt option was not specified.
ReqCountBlockLock	INTEGER	Returns the number of times the connection waited for a lock, or NULL if the -zt option was not specified.
ReqCountBlockContention	INTEGER	Returns the number of times the connection waited for atomic access, or NULL if the -zt option was not specified.
LastIdle	INTEGER	Returns the number of ticks between requests.
BlockedOn	INTEGER	Returns zero if the current connection isn't blocked, or if it is blocked, the connection number on which the connection is blocked because of a locking conflict.
UncommitOp	INTEGER	Returns the number of uncommitted operations.

Column Name	Data Type	Description
CurrentProcedure	VARCHAR(255)	Returns the name of the procedure that a connection is currently executing. If the connection is executing nested procedure calls, the name is the name of the current procedure. If there is no procedure executing, an empty string is returned.
EventName	VARCHAR(255)	Returns the name of the associated event if the connection is running an event handler. Otherwise, an empty string is returned.
CurrentLineNumber	INTEGER	Returns the current line number of the procedure or compound statement a connection is executing. The procedure can be identified using the CurrentProcedure property. If the line is part of a compound statement from the client, an empty string is returned.
LastStatement	LONG VARCHAR	Returns the most recently prepared SQL statement for the current connection. The LastStatement value is set when a statement is prepared, and is cleared when a statement is dropped. Only one statement string is remembered for each connection. If sa_conn_activity reports a non-empty value for a connection, it is most likely the statement that the connection is currently executing. If the statement had completed, it would likely have been dropped and the property value would have been cleared. If an application prepares multiple statements and retains their statement handles, then the LastStatement value does not reflect what a connection is currently doing. When client statement caching is enabled, and a cached statement is reused, this property returns an empty string.
LastPlanText	LONG VARCHAR	Returns the long text plan of the last query executed on the connection. You control the remembering of the last plan by setting the RememberLastPlan option of the sa_server_option system procedure, or using the -zp server option.

Column Name	Data Type	Description
AppInfo	LONG VARCHAR	Returns information about the client that made the connection. For HTTP connections, this includes information about the browser. For connections using older versions of jConnect or Sybase Open Client, the information may be incomplete. The API value can be DBLIB, ODBC, OLEDB, ADO.NET, iAnywhereJDBC, PHP, PerlDBD, or DBEXPRESS.
LockCount	INTEGER	Returns the number of locks held by the connection.
SnapshotCount	INTEGER	Returns the number of snapshots associated with the connection.

Remarks

The **sa_performance_diagnostics** system procedure returns a result set consisting of a set of request timing properties and statistics if the server has been told to collect the information. Recording of request timing information must be turned on the database server before calling **sa_performance_diagnostics**. To do this, specify the -zt option when starting the database server or execute the following:

```
CALL sa_server_option( 'RequestTiming','ON' );
```

Privileges

Requires the MONITOR system privilege.

Example

You can execute the following query to identify connections that have spent a long time waiting for database server requests to complete.

```
SELECT Number, Name,
      CAST( DATEDIFF( second, LoginTime, CURRENT TIMESTAMP ) AS
DOUBLE ) AS T,
      IF T <> 0 THEN (ReqTimeActive / T) ELSE NULL ENDIF AS
PercentActive
FROM sa_performance_diagnostics()
WHERE T > 0 AND PercentActive > 10.0
ORDER BY PercentActive DESC;
```

Find all requests that are currently executing, and have been executing for more than 60 seconds:

```
SELECT Number, Name,
       CAST( DATEDIFF( second, LastReqTime, CURRENT TIMESTAMP ) AS
DOUBLE ) AS ReqTime
FROM sa_performance_diagnostics()
WHERE ReqStatus <> 'IDLE' AND ReqTime > 60.0
ORDER BY ReqTime DESC;
```

sa_report_deadlocks System Procedure

Retrieves information about deadlocks from an internal buffer created by the database server.

Syntax

sa_report_deadlocks( )

Result set

Column Name	Data Type	Description
snapshotId	BIGINT	The deadlock instance (all rows pertaining to a particular deadlock have the same ID).
snapshotAt	TIMESTAMP	The time when the deadlock occurred.
waiter	INT	The connection handle of the waiting connection.
who	VARCHAR(128)	The user ID associated with the connection that is waiting.
what	LONG VARCHAR	The command being executed by the waiting connection. This information is only available if you have turned on capturing of the most recently-prepared SQL statement by specifying the **-zl** option on the database server command line.
object_id	UNSIGNED BIGINT	The object ID of the table containing the row.
record_id	BIGINT	The row ID for system tables
owner	INT	The connection handle of the connection owning the lock being waited on.
is_victim	BIT	Identifies the rolled back transaction.
rollback_operation_count	UNSIGNED INT	The number of uncommitted operations that may be lost if the transaction rolls back.

Column Name	Data Type	Description
iq_rid	UNSIGNED BIGINT	The row ID for IQ RLV enabled tables.
iq_txn_id	UNSIGNED BIGINT	The transaction id of the associated row

Remarks
When the log_deadlocks option is set to On, the database server logs information about deadlocks in an internal buffer. You can view the information in the log using the sa_report_deadlocks system procedure.

Privileges
You must have the MONITOR system privilege.

Side effects
None.

sa_rowgenerator system procedure

Returns a result set with rows between a specified start and end value.

Syntax

```
sa_rowgenerator(
 [ rstart
 [, rend
 [, rstep ] ] ]
)
```

Arguments

- ***rstart*** **–** Use this optional INTEGER parameter to specify the starting value. The default value is 0.
- ***rend*** **–** Use this optional INTEGER parameter to specify the ending value that is greater than or equal to *rstart*. The default value is 100.
- ***rstep*** **–** Use this optional INTEGER parameter to specify the increment by which the sequence values are increased. The default value is 1.

Result set

Column name	Data type	Description
row_num	INTEGER	Sequence number.

Remarks

The sa_rowgenerator procedure can be used in the FROM clause of a query to generate a sequence of numbers. This procedure is an alternative to using the RowGenerator system table. You can use sa_rowgenerator for such tasks as:

- generating test data for a known number of rows in a result set.
- generating a result set with rows for values in every range. For example, you can generate a row for every day of the month, or you can generate ranges of zip codes.
- generating a query that has a specified number of rows in the result set. This may be useful for testing the performance of queries.

No rows are returned if you do not specify correct start and end values and a positive non-zero step value.

You can emulate the behavior of the RowGenerator table with the following statement:

```
SELECT row_num FROM sa_rowgenerator( 1, 255 );
```

Privileges

None

Side effects

None

Example

The following query returns a result set containing one row for each day of the current month.

```
SELECT DATEADD( day, row_num-1,
        YMD( DATEPART( year, CURRENT DATE ),
            DATEPART( month, CURRENT DATE ), 1 ) )
    AS day_of_month
    FROM sa_rowgenerator( 1, 31, 1 )
   WHERE DATEPART( month, day_of_month ) = DATEPART( month, CURRENT
DATE )
    ORDER BY row_num;
```

The following query shows how many employees live in zip code ranges (0-9999), (10000-19999), ..., (90000-99999). Some of these ranges have no employees, which causes a warning.

The sa_rowgenerator procedure can be used to generate these ranges, even though no employees have a zip code in the range.

```
SELECT row_num AS r1, row_num+9999 AS r2, COUNT( PostalCode ) AS
zips_in_range
    FROM sa_rowgenerator( 0, 99999, 10000 ) D LEFT JOIN Employees
        ON PostalCode BETWEEN r1 AND r2
    GROUP BY r1, r2
    ORDER BY 1;
```

The following example generates 10 rows of data and inserts them into the NewEmployees table:

```
INSERT INTO NewEmployees ( ID, Salary, Name )
    SELECT row_num, CAST( RAND() * 1000 AS INTEGER ), 'Mary'
    FROM sa_rowgenerator( 1, 10 );
```

The following example uses the sa_rowgenerator system procedure to create a view containing all integers. The value 2147483647 in this example represents the maximum signed integer that is supported.

```
CREATE VIEW Integers AS
    SELECT row_num AS n
    FROM sa_rowgenerator( 0, 2147483647, 1 );
```

This example uses the sa_rowgenerator system procedure to create a view containing dates from 0001-01-01 to 9999-12-31. The value 3652058 in this example represents the number of days between 0001-01-01 and 9999-12-31, the earliest and latest dates that are supported.

```
CREATE VIEW Dates AS
    SELECT DATEADD( day, row_num, '0001-01-01' ) AS d
    FROM sa_rowgenerator( 0, 3652058, 1 );
```

The following query returns all years between 1900 and 2058 that have 54 weeks.

```
SELECT DATEADD ( day, row_num, '1900-01-01' ) AS d, DATEPART ( week,
d ) w
    FROM sa_rowgenerator ( 0, 63919, 1 )
    WHERE w = 54;
```

sa_server_option System Procedure

Overrides a server option while the server is running..

Syntax

sa_server_option(opt , val)

Arguments

- **opt –** Use this CHAR(128) parameter to specify a server option name.
- **val –** Use this CHAR(128) parameter to specify the new value for the server option.

Remarks

Option Name	Values	Additional information
AutoMultiProgrammingLevel	YES, NO	Default is YES. When set to YES, the database server automatically adjusts its multiprogramming level, which controls the maximum number of tasks that can be active at a time. If you choose to control the multiprogramming level manually by setting this option to NO, you can still set the initial, minimum, and maximum values for the multiprogramming level.

Option Name	Values	Additional information
AutoMultiProgrammingLevelStatistics	YES, NO	Default is NO. When set to YES, statistics for automatic multiprogramming level adjustments appear in the database server message log.
CacheSizingStatistics	YES, NO	Default is NO. When set to YES, display cache information in the database server messages window whenever the cache size changes.
CollectStatistics	YES, NO	Default is YES. When set to YES, the database server collects Performance Monitor statistics.
ConnsDisabled	YES, NO	Default is NO. When set to YES, no other connections are allowed to any databases on the database server.
ConnsDisabledForDB	YES, NO	Default is NO. When set to YES, no other connections are allowed to the current database.
ConsoleLogFile	filename	The name of the file used to record database server message log information. Specifying an empty string stops logging to the file. Double any backslash characters in the path because this value is a SQL string.
ConsoleLogMaxSize	file-size (bytes)	The maximum size, in bytes, of the file used to record database server message log information. When the database server message log file reaches the size specified by either this property or the **-on** server option, the file is renamed with the extension **.old** appended (replacing an existing file with the same name if one exists). The database server message log file is then restarted.
CurrentMultiProgrammingLevel	integer	Default is 20. Sets the multiprogramming level of the database server.
DatabaseCleaner	ON, OFF	Default is ON. Do not change the setting of this option except on the recommendation of Technical Support.

Option Name	Values	Additional information
DeadlockLog-ging	ON, OFF, RESET, CLEAR	Default is OFF. Controls deadlock logging. The value deadlock_logging is also sup-ported. The following values are supported: • **ON –** Enables deadlock logging. • **OFF –** Disables deadlock logging and leaves the deadlock data available for viewing. • **RESET –** Clears the logged deadlock data, if any exists, and then enables deadlock logging. • **CLEAR –** Clears the logged deadlock data, if any exists, and then disables deadlock logging. Once deadlock logging is enabled, you can use the sa_report_dead-locks system procedure to retrieve deadlock information from the database.
DebuggingInfor-mation	YES, NO	Default is NO. Displays diagnostic messages and other messages for troubleshoot-ing purposes. The messages appear in the database server messages window.
DiskSandbox	ON, OFF	Default is OFF. Sets the default disk sandbox settings for all databases started on the database server that do not have explicit disk sandbox settings. Changing the disk sandbox settings by using the sa_server_option system procedure does not affect databases already running on the database server. To use the sa_server_option system procedure to change disk sandbox settings, you must provide the secure feature key for the manage_disk_sandbox secure feature.
DropBadStatis-tics	YES, NO	Default is YES. Allows automatic statistics management to drop statistics that return bad estimates from the database.
DropUnusedSta-tistics	YES, NO	Default is YES. Allows automatic statistics management to drop statistics that have not been used for 90 consecutive days from the database.
IdleTimeout	Integer (mi-nutes)	Default is 240. Disconnects TCP/IP connections that have not submitted a request for the specified number of minutes. This prevents inactive connec-tions from holding locks indefinitely

Option Name	Values	Additional information
IPAddressMonitorPeriod	Integer (seconds)	The minimum value is 10 and the default is 0. For portable devices, the default value is 120. Sets the time to check for new IP addresses in seconds.
LivenessTimeout	Integer (seconds)	Default is 120. A liveness packet is sent periodically across a client/server TCP/IP network to confirm that a connection is intact. If the network server runs for a LivenessTimeout period without detecting a liveness packet, the communication is severed.
MaxMultiProgrammingLevel	Integer	Default is four times the value for CurrentMultiProgrammingLevel. Sets the maximum database server multiprogramming level.
MessageCategoryLimit	Integer	Default is 400. Sets the minimum number of messages of each severity and category that can be retrieved using the sa_server_messages system procedure.
MinMultiProgrammingLevel	Integer	Default is the minimum of the value of the -gtc server option and the number of logical CPUs on the computer.
OptionWatchAction	MESSAGE, ERROR	Default is MESSAGE. Specifies the action that the database server takes when an attempt is made to set an option in the list. When OptionWatchAction is set to MESSAGE, and an option specified by OptionWatchList is set, a message appears in the database server messages window indicating that the option being set is on the options watch list.When OptionWatchAction is set to ERROR, an error is returned indicating that the option cannot be set because it is on the options watch list. You can view the current setting for this property by executing `SELECT DB_PROPERTY( 'OptionWatchAction' );`

Option Name	Values	Additional information

OptionWatchList	Comma-separated list of database options	Specifies a comma-separated list of database options that you want to be notified about, or have the database server return an error for, when they are set. The string length is limited to 128 bytes. By default, it is an empty string. For example, the following command adds the automatic_timestamp, float_as_double, and tsql_hex_constant option to the list of options being watched:
		`CALL sa_server_option( 'OptionWatchList','automatic_timestamp, float_as_double,tsql_hex_constant' );`
		You can view the current setting for this property by executing:
		`SELECT DB_PROPERTY( 'OptionWatchList' );`

ProcedureProfiling	YES, NO, RESET, CLEAR	Default is NO.
ProfileFilterConn	connection-id	Instructs the database server to capture profiling information for a specific connection ID, without preventing other connections from using the database. When connection filtering is enabled, the value returned for **SELECT PROPERTY('ProfileFilterConn')** is the connection ID of the connection being monitored. If no ID has been specified, or if connection filtering is disabled, the value returned is -1.

Option Name	Values	Additional information
ProcessorAffinity	Comma-delimited list of processor numbers and/or ranges. The default is that all processors are used or the setting of the -gta option.	Instructs the database server which logical processors to use on Windows or Linux. Specify a comma-delimited list of processor numbers and/or ranges. If the lower endpoint of a range is omitted, then it is assumed to be zero. If the upper endpoint of a range is omitted, then it is assumed to be the highest CPU known to the operating system. The in_use column returned by the sa_cpu_topology system procedure contains the current processor affinity of the database server, and the in_use column indicates whether the database server is using a processor. Alternatively, you can query the value of the ProcessorAffinity database server property. The database server might not use all of the specified logical processors in the following cases: • If one or more of the specified logical processors does not exist, or is offline. • If the license does not allow it. If you specify an invalid processor ID, sa_server_option returns an error.
ProfileFilterUser	user-id	Instructs the database server to capture profiling information for a specific user ID.
QuittingTime	Valid date and time	Instructs the database server to shut down at the specified time.
RememberLastPlan	YES, NO	Default is NO. Instructs the database server to capture the long text plan of the last query executed on the connection. This setting is also controlled by the -zp server option.When RememberLastPlan is turned on, obtain the textual representation of the plan of the last query executed on the connection by querying the value of the LastPlanText connection property: `SELECT CONNECTION_PROPERTY( 'LastPlanText' );`

Option Name	Values	Additional information

RememberLast-Statement	YES, NO	Default is NO.
		
		Instructs the database server to capture the most recently prepared SQL statement for each database running on the server. For stored procedure calls, only the outermost procedure call appears, not the statements within the procedure.When RememberLastStatement is turned on, you can obtain the current value of the LastStatement for a connection by querying the value of the LastStatement connection property:
		
		`SELECT CONNECTION_PROPERTY( 'LastState-ment' );`
		
		When client statement caching is enabled, and a cached statement is reused, this property returns an empty string. When RememberLast-Statement is turned on, the following statement returns the most recently-prepared statement for the specified connection:
		
		`SELECT CONNECTION_PROPERTY( 'LastStatement', connection-id );`
		
		The sa_conn_activity system procedure returns this same informa-tion for all connections.
		
		Note: When -zl is specified, or when the RememberLastStatement server setting is turned on, any user can call the sa_conn_activity system procedure or obtain the value of the LastStatement connection property to find out the most recently-prepared SQL statement for any other user. Use this option with caution and turn it off when it is not required.

Option Name	Values	Additional information

RequestFilter-Conn	connec-tion-id, -1	Filter the request logging information so that only information for a particular connection is logged. This filtering can reduce the size of the request log file when monitoring a database server with many active connections or multiple databases. You can obtain the connection ID by executing the following:
		`CALL sa_conn_info( );`
		To log a specific connection once you have obtained the connection ID, execute the following statement:
		`CALL sa_server_option( 'RequestFilterConn', connection-id );`
		Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:
		`CALL sa_server_option( 'RequestFilterConn', -1 );`

RequestFilterDB	data-base-id, -1	Filter the request logging information so that only information for a particular database is logged. This can help reduce the size of the request log file when monitoring a server with multiple databases. You can obtain the database ID by executing the following statement when you are connected to the desired database:
		`SELECT CONNECTION_PROPERTY( 'DBNumber' );`
		To log only information for a particular database, execute the following statement:
		`CALL sa_server_option( 'RequestFilterDB', database-id );`
		Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:
		`CALL sa_server_option( 'RequestFilterDB', -1 );`

Option Name	Values	Additional information
RequestLogFile	filename	The name of the file used to record request information. Specifying an empty string stops logging to the request log file. If request logging is enabled, but the request log file was not specified or has been set to an empty string, the server logs requests to the database server messages window. Double any backslash characters in the path because this value is a SQL string. When client statement caching is enabled, set the **max_client_statements_cached** option to 0 to disable client statement caching while the request log is captured, if the log will be analyzed using the **tracetime.pl** Perl script.

Option Name	Values	Additional information
RequestLogging	SQL, HOST-VARS, PLAN, PROCE-DURES, TRIG-GERS, OTHER, BLOCKS, RE-PLACE, ALL, YES, NONE, NO	Default is NONE. This call turns on logging of individual SQL statements sent to the database server for use in troubleshooting with the database server -zr and -zo options. Values can be combinations of the following, separated by either a plus sign (+), or a comma: • **PLAN –** enables logging of execution plans (short form). If logging of procedures (PROCEDURES) is enabled, execution plans for procedures are also recorded. • **HOSTVARS –** enables logging of host variable values. If you specify HOSTVARS, the information listed for SQL is also logged. • **PROCEDURES –** enables logging of statements executed from within procedures. • **TRIGGERS –** enables logging of statements executed from within triggers. • **OTHER –** enables logging of additional request types not included by SQL, such as FETCH and PREFETCH. However, if you specify OTHER but do not specify SQL, it is the equivalent of specifying SQL+OTHER. Including OTHER can cause the log file to grow rapidly and could negatively impact server performance. • **BLOCKS –** enables logging of details showing when a connection is blocked and unblocked on another connection. • **REPLACE –** at the start of logging, the existing request log is replaced with a new (empty) one of the same name. Otherwise, the existing request log is opened and new entries are appended to the end of the file. • **ALL –** logs all supported information. This value is equivalent to specifying SQL+PLAN+HOSTVARS+PROCEDURES+TRIGGERS+OTHER+BLOCKS. This setting can cause the log file to grow rapidly and could negatively impact server performance. • **NO or NONE –** turns off logging to the request log. You can view the current setting for this property by executing: `SELECT PROPERTY( 'RequestLogging' );`

Option Name	Values	Additional information
RequestLogMax-Size	file-size (bytes)	The maximum size of the file used to record request logging information, in bytes. If you specify 0, then there is no maximum size for the request logging file, and the file is never renamed. This value is the default. When the request log file reaches the size specified by either the sa_server_option system procedure or the -zs server option, the file is renamed with the extension **.old** appended (replacing an existing file with the same name if one exists). The request log file is then restarted.
RequestLog-NumFiles	Integer	The number of request log file copies to retain.If request logging is enabled over a long period, the request log file can become large. The **-zn** option allows you to specify the number of request log file copies to retain
RequestTiming	YES, NO	Default is NO. Instructs the database server to maintain timing information for each new connection. This feature is turned off by default. When it is turned on, the database server maintains cumulative timers for all new connections that indicate how much time the connection spent in the server in each of several states. The change is only effective for new connections, and lasts for the duration each connection.You can use the sa_performance_diagnostics system procedure to obtain a summary of this timing information, or you can retrieve individual values by inspecting the following connection properties: • **ReqCountUnscheduled** • **ReqTimeUnscheduled** • **ReqCountActive** • **ReqTimeActive** • **ReqCountBlockIO** • **ReqTimeBlockIO** • **ReqCountBlockLock** • **ReqTimeBlockLock** • **ReqCountBlockContention** • **ReqTimeBlockContention** When the RequestTiming server property is on, there is a small overhead for each request to maintain the additional counters.

Option Name	Values	Additional information
rlv_auto_merge	ON, OFF	The default is ON. Enables or disables automatic merges of the RLV store into the IQ main store for row-level versioning-enabled tables. If rlv_auto_merge is OFF, no automated merges of the RLV and IQ main stores occur. This implies that you assume responsibility to manually merge data so that the RLV store gets synced to the IQ main store before the upper rlv_memory_mb threshold is reached.
rlv_memory_mb	The minimum value is 1 MB. The maximum value is 2048. Any other value will set the amount of memory to 2048 MB.	Specifies the maximum amount of memory (the RLV store), in MB, to reserve for row-level versioning . The default value is 2048 MB.

Option Name	Values	Additional information
SecureFeatures	feature-list	Allows you to manage secure features for a database server that is already running. The feature-list is a comma-separated list of feature names or feature sets. By adding a feature to the list, you limit its availability. To remove items from the list of secure features, specify a minus sign (-) before the secure feature name. To call sa_server_option('SecureFeatures',...), the connection must have the ManageFeatures secure feature enabled on the connection. The -sf key (the system secure feature key) enables ManageFeatures, as well as all of the other features. So if you used the system secure feature key, then changing the set of SecureFeatures will not have any effect on the connection. But if you used another key (for example a key that had been created using the create_secure_feature_key system procedure) then your connection may be immediately affected by the change, depending on what other features are included in the key. Any changes you make to allow or prevent access to features take effect immediately for the database server. The connection that executes the sa_server_option system procedure may or may not be affected, depending on the secure feature key the connection is using and whether or not it allows the connection access to the specified features. For example, to secure two features, use the following syntax: `CALL sa_server_option('SecureFeatures', 'CONSOLE_LOG,WEBCLIENT_LOG' );` After executing this statement, the list of secure features is set according to what has been changed.
StatisticsCleaner	ON, OFF	Default is ON. The statistics cleaner fixes statistics that give bad estimates by performing scans on tables. By default the statistics cleaner runs in the background and has a minimal impact on performance. Turning off the statistics cleaner does not disable the statistic governor, but when the statistics cleaner is turned off, statistics are only created or fixed when a query is run.
WebClientLogFile	filename	The name of the web service client log file. The web service client log file is truncated each time you use the -zoc server option or the WebClientLogFile property to set or reset the file name. Double any backslash characters in the path because this value is a string.

Option Name	Values	Additional information
WebClientLog-ging	ON, OFF	Default is OFF. This option enables and disables logging of web service clients. The information that is logged includes HTTP requests and response data. Specify ON to start logging to the web service client log file, and specify OFF to stop logging to the file.

Privileges
You must have the MANAGE PROFILING system privilege to use the following options, which are related to application profiling or request logging:

- ProcedureProfiling
- ProfileFilterConn
- ProfileFilterUser
- RequestFilterConn
- RequestFilterDB
- RequestLogFile
- RequestLogging
- RequestLogMaxSize
- RequestLogNumFiles

For all other options, your must have the SERVER OPERATOR system privilege.

Side effects
None.

Example
The following statement causes cache information to be displayed in the database server messages window whenever the cache size changes:

```
CALL sa_server_option( 'CacheSizingStatistics', 'YES' );
```

The following statement disallows new connections to the current database:

```
CALL sa_server_option( 'ConnsDisabledForDB', 'YES' );
```

The following statement enables logging of all SQL statements, procedure calls, plans, blocking and unblocking events, and starts a new request log:

```
CALL sa_server_option( 'RequestLogging', 'SQL+PROCEDURES+BLOCKS+PLAN
+REPLACE' );
```

sa_set_http_header system procedure

Permits a web service to set an HTTP response header.

Syntax

```
sa_set_http_header(
fldname
, val
)
```

Arguments

- ***fldname* –** Use this CHAR(128) parameter to specify a string containing the name of one of the HTTP header fields.
- ***val* –** Use this LONG VARCHAR parameter to specify the value to which the named parameter should be set. Setting a response header to NULL, effectively removes it.

Remarks

Setting the special header field @HttpStatus sets the status code returned with the request. The status code is also known as the response code. For example, the following script sets the status code to 404 Not Found:

```
CALL sa_set_http_header( '@HttpStatus', '404' );
```

You can create a user-defined status message by specifying a three digit status code with an optional colon-delimited text message. For example, the following script outputs a status code with the message "999 User Code":

```
CALL sa_set_http_header( '@HttpStatus', '999:User Code' );
```

Note: A user defined status text message is not translated into a database character-set when logged using the LogOptions protocol option.

The body of the error message is inserted automatically. Only valid HTTP error codes can be used. Setting the status to an invalid code causes a SQL error.

The sa_set_http_header procedure always overwrites the existing header value of the header field when called.

Response headers generated automatically by the database server can be removed. For example, the following command removes the Expires response header:

```
CALL sa_set_http_header( 'Expires', NULL );
```

Privileges

None

Side effects

None

Example

The following example sets the Content-Type header field to text/html.

```
CALL sa_set_http_header( 'Content-Type', 'text/html' );
```

sa_set_http_option system procedure

Permits a web service to set an HTTP option for process control.

Syntax

```
sa_set_http_option(
optname
, val
)
```

Arguments

- ***optname* –** Use this CHAR(128) parameter to specify a string containing the name of one of the HTTP options.

  The supported options are:
  - **CharsetConversion –** Use this option to control whether the result set is to be automatically converted from the character set encoding of the database to the character set encoding of the client. The only permitted values are ON and OFF. The default value is ON.
  - **AcceptCharset –** Use this option to specify the web server's preferences for a response character set encoding. One or more character set encodings may be specified in order of preference. The syntax for this option conforms to the syntax used for the HTTP Accept-Charset request-header field specification in RFC2616 Hypertext Transfer Protocol.

    An HTTP client such as a web browser may provide an Accept-Charset request header which specifies a list of character set encodings ordered by preference. Optionally, each encoding may be given an associated quality value (q=*qvalue*) which represents the client's preference for that encoding. By default, the quality value is 1 (q=1). Here is an example:

    ```
    Accept-Charset: iso-8859-5, utf-8;q=0.8
    ```

    A plus sign (+) in the AcceptCharset HTTP option value may be used as a shortcut to represent the current database character set encoding. The plus sign also indicates that the database character set encoding should take precedence if the client also specifies the encoding in its list, regardless of the quality value assigned by the client.

    An asterisk (*) in the AcceptCharset HTTP option may be used to indicate that the web service should use a character set encoding preferred by the client, as long as it is also supported by the server, when client and server do not have an intersecting list.

    When sending the response, the first character set encoding preferred by both client and web service is used. The client's order of preference takes precedence. If no mutual encoding preference exists, then the web service's most preferred encoding is used, unless an asterisk (*) appears in the web service list in which case the client's most preferred encoding is used.

If the AcceptCharset HTTP option is not used, the most preferred character set encoding specified by the client and supported by the server is used. If none of the encodings specified by the client are supported (or the client does not send an Accept-Charset request header) then the database character set encoding is used.

If a client does not send an Accept-Charset header then one of the following actions are taken:

- If the AcceptCharset HTTP option has not been specified then the web server will use the database character set encoding.
- If the AcceptCharset HTTP option has been specified then the web server will use its most preferred character set encoding.

If a client does send an Accept-Charset header then one of the following actions are taken:

- If the AcceptCharset HTTP option has not been specified then the web server will attempt to use one of the client's preferred character set encodings, starting with the most preferred encoding. If the web server does not support any of the client's preferred encodings, it will use the database character set encoding.
- If the AcceptCharset HTTP option has been specified then the web server will attempt to use the first preferred character set encoding common to both lists, starting with the client's most preferred encoding. For example, if the client sends an Accept-Charset header listing, in order of preference, encodings iso-a, iso-b, and iso-c and the web server prefers iso-b, then iso-a, and finally iso-c, then iso-a will be selected.

  ```
  Web client: iso-a, iso-b, iso-c
  Web server: iso-b, iso-a, iso-c
  ```

  If the intersection of the two lists is empty, then the web server's first preferred character set is used. From the following example, encoding iso-d will be used.

  ```
  Web client: iso-a, iso-b, iso-c
  Web server: iso-d, iso-e, iso-f
  ```

  If an asterisk ('*') was included in the AcceptCharset HTTP option, then emphasis would be placed on the client's choice of encodings, resulting in iso-a being used. Essentially, the use of an asterisk guarantees that the intersection of the two lists will not be empty.

The ideal situation occurs when both client and web service use the database character set encoding since this eliminates the need for character set translation and improves the response time of the web server.

    If the CharsetConversion option has been set to OFF, then AcceptCharset processing is not performed.

  - **SessionID –** Use this option to create, delete or rename an HTTP session. The database connection is persisted when a web service sets this option to create an HTTP session but sessions are not persisted across server restarts. If already within a session context, this call will rename the session to the new session ID. When called with a NULL value, the session will be deleted when the web service terminates.

    The generated session keys are limited to 128 characters in length and unique across databases if multiple databases are loaded.
  - **SessionTimeout –** Use this option to specify the amount of time, in minutes, that the HTTP session persists during inactivity. This time-out period is reset whenever an HTTP request uses the given session. The session is automatically deleted when the SessionTimeout is exceeded.
- ***val* –** Use this LONG VARCHAR parameter to specify the value to which the named option should be set.

Remarks

Use this procedure within statements or procedures that handle web services to set options.

When sa_set_http_option is called from within a procedure invoked through a web service, and either the option or option value is invalid, an error is returned.

Privileges

None

Side effects

None

Examples

The following example illustrates the use of sa_set_http_option to indicate the web service's preference for database character set encoding. The UTF-8 encoding is specified as a second choice. The asterisk (*) indicates that the web service is willing to use the character set encoding most preferred by the client, provided that it is supported by the web server.

```
CALL sa_set_http_option( 'AcceptCharset', '+,UTF-8,*');
```

The following example illustrates the use of sa_set_http_option to correctly identify the character encoding in use by the web service. In this example, the web server is connected to a 1251CYR database and is prepared to serve HTML documents containing the Cyrillic alphabet to any web browser.

```
CREATE OR REPLACE PROCEDURE cyrillic_html()
RESULT (html_doc XML)
BEGIN
  DECLARE pos INT;
  DECLARE charset VARCHAR(30);
  CALL sa_set_http_option( 'AcceptCharset', 'iso-8859-5, utf-8' );
  SET charset = CONNECTION_PROPERTY( 'CharSet' );
  -- Change any IANA labels like ISO_8859-5:1988
  -- to ISO_8859-5 for Firefox.
  SET pos = LOCATE( charset, ':' );
  IF pos > 0 THEN
    SET charset = LEFT( charset, pos - 1 );
  END IF;
  CALL sa_set_http_header( 'Content-Type', 'text/html; charset=' ||
        charset );
  SELECT  '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">' ||
    XMLCONCAT(
      XMLELEMENT('HTML',
        XMLELEMENT('HEAD',
          XMLELEMENT('TITLE', 'Cyrillic characters')
        ),
        XMLELEMENT('BODY',
          XMLELEMENT('H1', 'First 5 lowercase Russian letters'),
          XMLELEMENT('P', UNISTR('\u0430\u0431\u0432\u0433\u0434'))
        )
      )
    );
END;

CREATE SERVICE cyrillic
TYPE 'RAW'
AUTHORIZATION OFF
USER DBA
AS CALL cyrillic_html();
```

To illustrate the process of establishing the correct character set encoding to use, consider the following Accept-Charset header delivered by a web browser such as Firefox to the web service. It indicates that the browser prefers ISO-8859-1 and UTF-8 encodings but is willing to accept others.

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

The web service will not accept the ISO-8859-1 character set encoding since the web page to be transmitted contains Cyrillic characters. The web service prefers ISO-8859-5 or UTF-8 encodings as indicated by the call to sa_set_http_option. In this example, the UTF-8 encoding will be chosen since it is agreeable to both parties. The database connection property CharSet indicates which encoding has been selected by the web service. The sa_set_http_header procedure is used to indicate the HTML document's encoding to the web browser.

```
Content-Type: text/html; charset=UTF-8
```

If the web browser does not specify an Accept-Charset, then the web service defaults to its first preference, ISO-8859-5. The sa_set_http_header procedure is used to indicate the HTML document's encoding.

```
Content-Type: text/html; charset=ISO_8859-5
```

The following example sets a unique HTTP session identifier:

```
BEGIN
  DECLARE sessionid VARCHAR(30);
  DECLARE tm TIMESTAMP;
  SET tm = NOW(*);
  SET sessionid = 'MySessions_' ||
     CONVERT( VARCHAR, SECONDS(tm)*1000 + DATEPART(millisecond,tm));
  SELECT sessionid;
  CALL sa_set_http_option('SessionID', sessionid);
END;
```

The following example sets the time-out for an HTTP session to 5 minutes:

```
CALL sa_set_http_option('SessionTimeout', '5');
```

sa_table_page_usage system procedure

Reports information about the page usage of database tables.

Syntax

```
sa_table_page_usage( )
```

Result set

Column name	Data type	Description
TableId	UNSIGNED INTEGER	The table ID.
TablePages	INTEGER	The number of table pages used by the table.
PctUsedT	INTEGER	The percentage of used table page space.
IndexPages	INTEGER	The number of index pages used by the table.
PctUsedI	INTEGER	The percentage of used index page space.
PctOfFile	INTEGER	The percentage of the total database file the table occupies.
TableName	CHAR(128)	The table name.

Remarks

The results include the same information provided by the Information utility. When the progress_messages database option is set to Raw or Formatted, progress messages are sent from the database server to the client while the sa_table_page_usage system procedure is running.

Privileges

You must have the MANAGE ANY DBSPACE system privilege.

Side effects

None

Example

The following example obtains information about the page usage of the SalesOrderItems table.

```
SELECT * FROM sa_table_page_usage( )
    WHERE TableName = 'SalesOrderItems';
```

sa_text_index_stats System Procedure

Returns statistical information about the **TEXT** indexes in the database.

Syntax

```
sa_text_index_stats( )
```

Description

Use **sa_text_index_stats** to view statistical information for each **TEXT** index in the database.

Table 147. Statistical information for TEXT indexes returned by sa_text_index_stats

Column name	Type	Description
owner_id	UNSIGNED INT	ID of the owner of the table
table_id	UNSIGNED INT	ID of the table
index_id	UNSIGNED INT	ID of the **TEXT** index
text_con-fig_id	UNSIGNED BIGINT	ID of the text configuration referenced by the **TEXT** index
owner_name	CHAR(128)	Name of the owner
table_name	CHAR(128)	Name of the table
index_name	CHAR(128)	Name of the **TEXT** index
text_con-fig_name	CHAR(128)	Name of the text configuration object
doc_count	UNSIGNED BIGINT	Total number of indexed column values in the **TEXT** index

Column name	Type	Description
doc_length	UNSIGNED BIGINT	Total length of data in the **TEXT** index
pend-ing_length	UNSIGNED BIGINT	Total length of the pending changes
de-leted_length	UNSIGNED BIGINT	Total length of the pending deletions
last_refresh	TIMESTAMP	Date and time of the last refresh

The `pending_length`, `deleted_length`, and `last_refresh` values are NULL for **IMMEDIATE REFRESH TEXT** indexes.

Permissions
Any of the following system privileges:

- MANAGE ANY STATISTICS
- CREATE ANY INDEX
- ALTER ANY INDEX
- DROP ANY INDEX
- CREATE ANY OBJECT
- ALTER ANY OBJECT
- DROP ANY OBJECT

Example
Return statistical information for each **TEXT** index in the database:

```
CALL sa_text_index_stats( );
```

sa_text_index_vocab System Procedure

Lists all terms that appear in a **TEXT** index, and the total number of indexed values in which each term appears.

Syntax

```
sa_text_index_vocab (
    'text-index-name',
    'table-name',
    'table-owner'
    )
```

Parameters
text-index-name – use this `CHAR`(128) parameter to specify the name of the **TEXT** index.

table-name – use this `CHAR`(128) parameter to specify the name of the table on which the **TEXT** index is built.

table-owner – use this `CHAR`(128) parameter to specify the owner of the table.

Description

sa_text_index_vocab returns all terms that appear in a **TEXT** index, and the total number of indexed values in which each term appears (which is less than the total number of occurrences, if the term appears multiple times in some indexed values).

Parameter values cannot be host variables or expressions. The arguments *text-index-name*, *table-name*, and *table-owner* must be constraints or variables.

Privileges

SELECT ANY TABLE system privilege or SELECT permission on the indexed table is required.

Example

Execute **sa_text_index_vocab** to return all the terms that appear in the **TEXT** index `MyTextIndex` on table `Customers` owned by `GROUPO`:

```
sa_text_index_vocab
('MyTextIndex','Customers','GROUPO');
```

Table 148. Terms in the index

term	freq
a	1
Able	1
Acres	1
Active	5
Advertising	1
Again	1
...	...

sa_validate system procedure

Performs a checksum validation on all, or parts, of a database.

Syntax

```
sa_validate(
[ tbl_name
[, owner_name ] ]
)
```

Arguments

- ***tbl_name*** **–** Use this optional CHAR(128) parameter to specify the name of a table or materialized view to validate. The default is NULL.
- ***owner_name*** **–** Use this optional CHAR(128) parameter to specify an owner. When specified by itself, all tables and materialized views owned by the owner are validated. The default is NULL.

Privileges

You must have the VALIDATE ANY OBJECT system privilege.

Side effects

None

Remarks

Argument specified	Type of validation
None	All tables, materialized views, and indexes in the database are validated. The database itself is also validated, including checksum validation.
tbl_name	The specified table, or materialized view, and all of its indexes, that are owned by the current user are validated.
owner_name	All tables, materialized views, and indexes owned by the specified user are validated.
tbl_name and *owner_name*	The specified table, or materialized view, and all of its indexes, that are owned by the specified user are validated.

The procedure returns a single column named Messages. Errors returned during validation appear in the column. If validation succeeds without error, the column contains `No error detected`.

Warning! Validating a table or an entire database should be performed while no connections are making changes to the database; otherwise, errors may be reported indicating some form of database corruption even though no corruption actually exists.

Example

The following statement performs a validation of tables and materialized views owned by user DBA:

```
CALL sa_validate( owner_name = 'DBA' );
```

sa_verify_password system procedure

Validates the password of the current user.

Syntax

```
sa_verify_password( curr_pswd )
```

Arguments

- ***curr_pswd* –** Use this CHAR(128) parameter to specify the password of the current database user.

Returns

The function returns an INTEGER value.

Remarks

This procedure is used by sp_password. If the password matches, 0 is returned and no error occurs. If the password does not match, an error is diagnosed. The connection is not terminated if the password does not match.

Privileges

None

Side effects

None

Example

The following example attempts to validate the current connection's password when the current user is DBA or User1. An error occurs if the current password does not match.

```
IF USER_NAME() = 'DBA' THEN
    SELECT sa_verify_password( 'sql' );
ELSEIF USER_NAME() = 'User1' THEN
    SELECT sa_verify_password( 'user' );
END IF;
```

sp_alter_secure_feature_key System Procedure

Alters a previously-defined secure feature key by modifying the authorization key and/or the feature list.

Syntax

```
sp_alter_secure_feature_key (
    name,
    auth_key,
    features )
```

Arguments

- **name –** the VARCHAR (128) name for the secure feature key you want to alter. A key with the given name must already exist.
- **auth_key –** the CHAR (128) authorization key for the secure feature key. The authorization key must be either a non-empty string of at least six characters, or NULL, indicating that the existing authorization key is not to be changed.
- **features –** the LONG VARCHAR, comma-separated list of secure features that the key can enable. The feature_list can be NULL, indicating that the existing feature_list is not to be changed.

Remarks
This procedure allows you to alter the authorization key or feature list of an existing secure feature key.

Privileges
To use this procedure, you must be the database server owner and have the manage_keys feature enabled on the connection.

sp_auth_sys_role_info System Procedure

Generates a report which maps authorities to corresponding system roles and role id. This procedure returns a row for each authority.

Syntax
sp_auth_sys_role_info()

Result Set

Column Name	Data Type	Description
auth	varchar(20)	The name of the authority.
role_name	char(128)	The name of the equivalent system role.
role_id	unsigned int	The id of the system role.

Privileges
none

sp_create_secure_feature_key System Procedure

Creates a new secure feature key.

Syntax

```
sp_create_secure_feature_key (
    name,
    auth_key,
    features )
```

Arguments

- **name –** the VARCHAR (128) name for the new secure feature key. This argument cannot be NULL or an empty string.
- **auth_key –** the CHAR (128) authorization key for the secure feature key. The authorization key must be a non-empty string of at least six characters.
- **features –** the LONG VARCHAR comma-separated list of secure features that the new key can enable. Specifying "-" before a feature means that the feature is not re-enabled when the secure feature key is set.

Remarks

This procedure creates a new secure feature key that can be given to any user. The system secure feature key is created using the -sk database server option.

Privileges

To use this procedure, you must be the database server owner and have the manage_keys feature enabled on the connection.

sp_displayroles System Procedure

Displays all roles granted to a user-defined role or a user, or displays the entire hierarchical tree of roles.

Syntax

```
dbo.sp_displayroles(
 [user_role_name],
 [display_mode],
 [grant_type] )
```

Arguments

- **user_role_name –** valid values are:
  - A valid system privilege name or system privilege role name
  - A valid user-defined role name
  - A valid user name

By default, if no argument is specified, the current login user is used.

- **display_mode –** valid values are:
  - **EXPAND_UP** – shows all roles granted the input role or system privilege; that is the role hierarchy tree for the parent levels.
  - **EXPAND_DOWN** – shows all roles or system privileges granted to the input role or user; that is, the role hierarchy tree for the child levels,

  If no argument is specified (default), only the directly granted roles or system privileges appear.
- **grant_type –** valid values are:
  - **ALL** – shows all roles or system privileges granted.
  - **NO_ADMIN** – shows all roles or system privileges granted with the WITH NO ADMIN OPTION or WITH ADMIN OPTION clause.
  - **ADMIN** – shows all roles or system privileges granted with the WITH ADMIN OPTION or WITH ADMIN ONLY OPTION clause.

  If no argument is specified, **ALL** is used.

Result Set

Column Name	Data Type	Description
role_name	char(128)	Lists role/system privilege name.
parent_role_name	char(128)	Lists role name of the parent.
grant_type	char(10)	Lists grant type.
role_level	smallint	For `Expand_down` mode, 1 indicates directly granted roles; 2 indicates the next hierarchy below, and so on. For `Expand_up` mode, 0 indicates the roles to which the specified role is granted; -1 indicates the next hierarchy above, and so on.

Remarks

For Name = System privilege name, the results show the system privilege name instead of the system privilege role name.

For Mode = Expand_down, parent_role_name is NULL for level 1 (directly granted roles). If no mode is specified (default), role_level is 1 and parent_role_name is NULL, since only directly granted roles appear.

For Name = User name, with Mode = expand_up, no results are returned since a user resides at the top level in any role hierarchy. Similarly, if Name = an immutable system privilege name, with Mode = Expand_down, no results are returned because an immutable system privilege resides at the bottom level in any role hierarchy.

For default Mode, parent_role_name column is NULL and role_level is 1.

Privileges

For users and extended users:

- No system privilege is required to execute this procedures against themselves.
- MANAGE ROLES system privilege is required to execute this procedure for other users.

For roles and system privileges:

- Administrative privilege over the role or system privilege is required to execute this procedure.
- Role administrators can execute this procedure for roles they administer.

Example

This example assumes these GRANT statements have been executed:

```
GRANT SERVER OPERATOR TO r4;
GRANT BACKUP DATABASE TO r3 WITH ADMIN OPTION;
GRANT DROP CONNECTION TO r3 WITH ADMIN ONLY OPTION;
GRANT MONITOR TO r2;GRANT CHECKPOINT TO r1;
GRANT ROLE r2 TO r1 WITH ADMIN OPTION;
GRANT ROLE r3 TO r2 WITH NO ADMIN OPTION;
GRANT ROLE r4 TO r3 WITH ADMIN ONLY OPTION;
GRANT ROLE r1 TO user1;
GRANT ROLE r1 TO r7;
GRANT ROLE r7 TO user2 WITH ADMIN OPTION;
GRANT BACKUP DATABASE TO user2 WITH ADMIN ONLY OPTION;
```

`sp_displayroles( 'user2', 'expand_down', 'ALL' )` produces output similar to:

role_name	pa-rent_role_name	grant_type	role_level
r7	NULL	ADMIN	1
PUBLIC	NULL	NO ADMIN	1
BACKUP DATABASE	NULL	ADMIN ONLY	1
dbo	PUBLIC	NO ADMIN	2
r1	r7	NO ADMIN	2
r2	r1	ADMIN	3

role_name	parent_role_name	grant_type	role_level
CHECKPOINT	r1	NO ADMIN	3
r3	r2	NO ADMIN	4
MONITOR	r2	NO ADMIN	4
r4	r3	ADMIN ONLY	5
BACKUP DATABASE	r3	ADMIN	5
DROP CONNECTION	r3	ADMIN ONLY	5

`sp_displayroles( 'user2', 'expand_down', 'NO_ADMIN' )` produces output similar to:

role_name	parent_role_name	grant_type	role_level
r7	NULL	ADMIN	1
PUBLIC	NULL	NO ADMIN	1
dbo	PUBLIC	NO ADMIN	2
r1	r7	NO ADMIN	2
r2	r1	ADMIN	3
CHECKPOINT	r1	NO ADMIN	3
r3	r2	NO ADMIN	4
MONITOR	r2	NO ADMIN	4
BACKUP DATABASE	r3	ADMIN	5

`sp_displayroles( 'r3', 'expand_up', 'NO_ADMIN' )` produces out put similar to:

role_name	parent_role_name	grant_type	role_level
r1	r7	NO ADMIN	-2
r2	r1	ADMIN	-1
r3	r2	NO ADMIN	0

`sp_displayroles( 'r1', 'NO_ADMIN', 'expand_up')` produces output similar to:

role_name	pa-rent_role_name	grant_type	role_level
r1	r7	NO ADMIN	0

sp_drop_secure_feature_key System Procedure

Deletes a secure feature key.

Syntax

```
sp_drop_secure_feature_key ( name )
```

Arguments

- **name –** the VARCHAR (128) name of the secure feature key to drop.

Remarks

If the named key does not exist, an error is returned. If the named key exists, it is deleted as long as it is not the last secure feature key that is allowed to manage secure features and secure feature keys. For example, the system secure feature key cannot be dropped until there is another key that has the manage_features and manage_keys secure features enabled.

Privileges

To use this procedure, you must be the database server owner and have the manage_keys feature enabled on the connection.

sp_expireallpasswords system procedure

Immediately expires all user passwords.

Syntax1

```
call sp_expireallpasswords
```

Syntax2

```
sp_expireallpasswords
```

Privileges

Requires the MANAGE ANY USER system privilege.

See also

- *sp_iqaddlogin Procedure* on page 429
- *sp_iqcopyloginpolicy Procedure* on page 459
- *sp_iqmodifylogin Procedure* on page 519

- *sp_iqpassword Procedure* on page 534

sp_list_secure_feature_keys System Procedure

Returns information about the contents of a directory.

Syntax

```
sp_list_secure_feature_keys ( )
```

Result Set

Column Name	Data Type	Description
name	VARCHAR(128)	The name of the secure feature key.
features	LONG VARCHAR	The secure features enabled by the secure feature key.

Remarks

This procedures returns the names of existing secure feature keys, as well as the set of secure features that can be enabled by each key.

If the user has the manage_features and manage_keys secure features enabled, then the procedure returns a list of all secure feature keys.

If the user only has the manage_keys secure feature enabled, then the procedure returns keys that have the same features or a subset of the same features that the current user has enabled.

Privileges

To use this procedure, you must be the database server owner and have the manage_keys feature enabled on the connection.

sp_login_environment system procedure

Sets connection options when users log in.

Syntax

```
sp_login_environment( )
```

Remarks

sp_login_environment is the default procedure called by the login_procedure database option.

It is recommended that you do not edit this procedure. Instead, to change the login environment, set the login_procedure option to point to a different procedure.

Privileges

None

Side effects

None

sp_objectpermission System Procedure

Generates a report on object permissions granted to the specified role, or user name, or the object permissions granted on the specified object or dbspace.

Syntax

```
dbo.sp_objectpermission (
[object_name],
[object_owner],
[object_type] )
```

Arguments

Arguments	Description
object_name	The name of an object or dbspace or a user or a role. If not specified, object permissions of the current user are reported. Default value is NULL.
object_owner	The name of the object owner for the specified object name. The object permissions of the specified object owned by the specified object owner are displayed. This parameter must be specified to obtain the object permissions of an object owned by another user or role. Default value is NULL.
object_type	Valid values are: • TABLE* • VIEW • MATERIALIZED VIEW • SEQUENCE • PROCEDURE • FUNCTION • DBSPACE • USER **Note:** *Column-level object permissions also appear. If no value is specified, permissions on all object types are returned. Default value is NULL.

Result Set

Column Name	Data Ttype	Description
grantor	char(128)	The user ID of the grantor

Column Name	Data Ttype	Description
grantee	char(128)	The user ID of the grantee
object_name	char(128)	The name of the object
object_type	char(20)	The type of object
column_name	char(128)	The name of the column
permission	char(20)	The name of the permission
grantable	char(1)	Whether or not the permission is grantable

Remarks

All arguments are optional and can generate these reports:

- If input is an object (table, view, procedure, function, sequence, and so on), procedure displays list of all roles and user that have different object permission on the object.
- If input is a role or user, procedure displays list of all object privileges granted to the role or input. When executing **sp_objectpermission** to display object permissions of a user or a role, the object permissions that are inherited through role grants also.
- If input is a dbspace name, procedure displays list of all user or roles that have CREATE permission on the specified dbspace.
- By default, object type is NULL and the object permissions for all existing object types matching the specified object name appear.

Privileges

- Any user can execute **sp_objectpermission** to obtain all the object permissions granted to him- or herself,
- Object owners can execute **sp_objectpermission** to obtain the object permissions for self-owned objects.
- MANAGE ANY OBJECT PRIVILEGE system privilege is required to obtain object permissions that are granted:
  - On objects owned by other users
  - To other users

  .
- MANAGE ANY OBJECT PRIVILEGE system privilege or role administrator is required to obtain object permissions that are granted:
  - On objects owned by a role
  - To a role.
- MANAGE ANY DBSPACE system privilege required to obtain permissions of a dbspace.

Example

The following GRANT statements are executed:

```
GRANT SERVER OPERATOR TO r4;
GRANT BACKUP DATABASE TO r3 WITH ADMIN OPTION;
GRANT DROP CONNECTION TO r3 WITH ADMIN ONLY OPTION;
GRANT MONITOR TO r2;GRANT CHECKPOINT TO r1;
GRANT ROLE r2 TO r1 WITH ADMIN OPTION;
GRANT ROLE r3 TO r2 WITH NO ADMIN OPTION;
GRANT ROLE r4 TO r3 WITH ADMIN ONLY OPTION;
```

Consider these object permissions:

- `r5` owns a table named `test_tab` and a procedure named `test_proc` in the database.
- `u5`, which has administrative rights over `r5`, grants the following permissions:
  - GRANT SELECT ON r5.test_tab TO r2 WITH GRANT OPTION;
  - GRANT SELECT (c1), UPDATE (c1) ON r5.test_tab TO r6 WITH GRANT OPTION;
  - GRANT EXECUTE ON r5.test_proc TO r3;
- `u6`, which has administrative rights over `r6`, grants the following permissions:
  - GRANT SELECT (c1), REFERENCES (c1) ON r5.test_tab TO r3;

If `sp_objectpermission( ‘r1’ )` is executed, output is similar to:

grantor	**grantee**	**object_name**	**owner**	**object_type**	**column_name**	**permission**	**grantable**
u5	r2	test_tab	r5	TABLE	NULL	SELECT	Y
u6	r3	test_tab	r5	COLUMN	c1	SELECT	N
u6	r3	test_tab	r5	COLUMN	c1	REFERENCES	N
u6	r3	test_proc	r5	PROCEDURE	NULL	EXECUTE	N

If `sp_objectpermission( ‘test_tab’, ‘r5’, ‘table’ )` is executed, output is similar to:

granter	**grantee**	**object_name**	**owner**	**object_type**	**column_name**	**permission**	**grantable**
u5	r2	test_tab	r5	TABLE	NULL	SELECT	Y
u5	r6	test_tab	r5	COLUMN	c1	SELECT	Y

granter	grantee	object_name	owner	object_type	column_name	permission	grantable
u5	r6	test_tab	r5	COLUMN	c1	UPDATE	Y
u6	r3	test_tab	r5	COLUMN	c1	SELECT	N
u6	r3	test_tab	r5	COLUMN	c1	REFERENCES	N

sp_proc_priv system procedure

Generates a report of the minimum system privileges required to run a stored procedure and pass the permission check for the procedure.

Syntax

sp_proc_priv ([*proc_name*])

Result set

Column name	Data type	Description
proc_name	char(128)	The name of the stored procedure.
privilege	long varchar	The privileges required to pass permission check.

Remarks

If multiple system privileges, separated by a comma, are displayed for a stored procedure, this implies that any one of them would suffice to execute the stored procedure. If multiple rows are displayed for a stored procedure, then one system privilege from each row is required to execute the stored procedure.

This procedure lists only those system privileges for a stored procedure that will always pass the permission check for the procedure. There may be other system privileges which would pass the permission check to execute the procedure given conditions, but these are not listed by this procedure.

Privileges

none

Example 1

If **sp_proc_priv** is invoked without any parameter specified, the procedure displays all the stored procedures and the system privileges required to execute each. Stored procedures which do not require any system privileges for their execution are not displayed.

If **sp_proc_priv ()** is executed, output would be similar to the following:

proc_name	privileges
sp_iqrowdensity	MONITOR, MANAGE ANY DBSPACE, CREATE ANY INDEX, ALTER ANY INDEX, CREATE ANY OBJECT, ALTER ANY OBJECT
sp_iqworkmon	MONITOR
sp_iqindexsize	MANAGE ANY DBSPACE, ALTER ANY INDEX, ALTER ANY OBJECT
sp_addlogin	MANAGE ANY USER
sp_iqemptyfile	BACKUP DATABASE, SERVER OPERATOR, ALTER DATABASE
sp_iqemptyfile	INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE, ALTER ANY TABLE, LOAD ANY TABLE, TRUNCATE ANY TABLE, ALTER ANY OBJECT
...	...

If **sp_proc_priv** is invoked with a procedure name parameter, it returns the system privileges required to execute that procedure. If no system privileges are required, it lists “No Privilege Required” against the procedure.

proc_name	privileges
sp_iqindexsize	MANAGE ANY DBSPACE, ALTER ANY INDEX

An error message appears if the procedure does not exist.

sp_remote_columns system procedure

Produces a list of the columns in a remote table, and a description of their data types.

The server must be defined with the CREATE SERVER statement to use this system procedure.

Syntax

```
sp_remote_columns(
@server_name
, @table_name
[, @table_owner
[, @table_qualifier ] ]
)
```

Arguments

- ***@server_name* –** Use this CHAR(128) parameter to specify a string containing the server name as specified by the CREATE SERVER statement.
- ***@table_name* –** Use this CHAR(128) parameter to specify the name of the remote table.
- ***@table_owner* –** Use this optional CHAR(128) parameter to specify the owner of *table_name*. The default is '%'.
- ***@table_qualifier* –** Use this optional CHAR(128) parameter to specify the name of the database in which *table_name* is located. The default is '%'.

Result set

Column name	Data type	Description
database	CHAR(128)	The database name.
owner	CHAR(128)	The database owner name.
table_name	CHAR(128)	The table name.
column_name	CHAR(128)	The name of a column.
domain_id	SMALLINT	An INTEGER which indicates the data type of the column.
width	INTEGER	The meaning of this column depends on the data type. For character types width represents the number of characters.
scale	SMALLINT	The meaning of this column depends on the data type. For NUMERIC data types scale is the number of digits after the decimal point.
nullable	SMALLINT	If NULL column values are allowed, the value is 1. Otherwise the value is 0.
base_type_str	CHAR(4096)	The annotated type string representing the physical type of the column.

Remarks

If you are entering a CREATE EXISTING TABLE statement and you are specifying a column list, it may be helpful to get a list of the columns that are available on a remote table. sp_remote_columns produces a list of the columns on a remote table and a description of their data types. If you specify a database, you must either specify an owner or provide the value NULL.

Privileges

None

Side effects

None

Standards and compatibility

- **Sybase –** Supported by Open Client/Open Server.

Example

The following example returns information about the columns in the ULProduct table in the remote SAP Sybase IQ database server named RemoteSA. The table owner is DBA.

```
CALL sp_remote_columns( 'RemoteSA', 'ULProduct', 'DBA', null );
```

The following example returns information about the columns in the SYSOBJECTS table in the Adaptive Server Enterprise database Production using the remote server named RemoteASE. The table owner is unspecified.

```
CALL sp_remote_columns( 'RemoteASE', 'sysobjects', null,
'Production' );
```

The following example returns formation about the columns in the Customers table in the Microsoft Access database `c:\users\me\documents\MyAccesDB.accdb` using the remote server MyAccessDB. The Access database does not have a table owner so NULL is specified.

```
CALL sp_remote_columns( 'MyAccessDB', 'Customers', null, 'c:\\users\
\me\\documents\\MyAccesDB.accdb' );
```

sp_remote_exported_keys system procedure

Provides information about tables with foreign keys on a specified primary table.

The server must be defined with the CREATE SERVER statement to use this system procedure.

Syntax

```
sp_remote_exported_keys(
@server_name
, @table_name
[, @table_owner
[, @table_qualifier ] ]
)
```

Arguments

- ***@server_name* –** Use this CHAR(128) parameter to specify the server the primary table is located on.
- ***@table_name* –** Use this CHAR(128) parameter to specify the table containing the primary key.
- ***@table_owner* –** Use this optional CHAR(128) parameter to specify the primary table's owner. The default is '%'.
- ***@table_qualifier* –** Use this optional CHAR(128) parameter to specify the database containing the primary table. The default is '%'.

Result set

Column name	Data type	Description
pk_database	CHAR(128)	The database containing the primary key table.
pk_owner	CHAR(128)	The owner of the primary key table.
pk_table	CHAR(128)	The primary key table.
pk_column	CHAR(128)	The name of the primary key column.
fk_database	CHAR(128)	The database containing the foreign key table.
fk_owner	CHAR(128)	The foreign key table's owner.
fk_table	CHAR(128)	The foreign key table.
fk_column	CHAR(128)	The name of the foreign key column.
key_seq	SMALLINT	The key sequence number.
fk_name	CHAR(128)	The foreign key name.
pk_name	CHAR(128)	The primary key name.

Remarks

This procedure provides information about the remote tables that have a foreign key on a particular primary table. The result set for the sp_remote_exported_keys system procedure includes the database, owner, table, column, and name for both the primary and the foreign key, and the foreign key sequence for the foreign key columns. The result set may vary because of the underlying ODBC and JDBC calls, but information about the table and column for a foreign key is always returned.

Privileges
None

Side effects
None

Example

This example returns information about the foreign key relationships in the ULEmployee table on the remote server named RemoteSA:

```
CALL sp_remote_exported_keys( 'RemoteSA', 'ULEmployee', 'DBA' );
```

sp_remote_imported_keys system procedure

Provides information about remote tables with primary keys that correspond to a specified foreign key.

The server must be defined with the CREATE SERVER statement to use this system procedure.

Syntax

```
sp_remote_imported_keys(
@server_name
, @table_name
[, @table_owner
[, @table_qualifier ] ]
)
```

Arguments

- ***@server_name* –** Use this CHAR(128) parameter to specify the server the foreign key table is located on. A value is required for this parameter.
- ***@table_name* –** Use this CHAR(128) parameter to specify the table containing the foreign key. A value is required for this parameter.
- ***@table_owner* –** Use this optional CHAR(128) parameter to specify the foreign key table's owner. The default is '%'.
- ***@table_qualifier* –** Use this optional CHAR(128) parameter to specify the database containing the foreign key table. The default is '%'.

Result set

Column name	Data type	Description
pk_database	CHAR(128)	The database containing the primary key table.
pk_owner	CHAR(128)	The owner of the primary key table.
pk_table	CHAR(128)	The primary key table.
pk_column	CHAR(128)	The name of the primary key column.
fk_database	CHAR(128)	The database containing the foreign key table.
fk_owner	CHAR(128)	The foreign key table's owner.
fk_table	CHAR(128)	The foreign key table.
fk_column	CHAR(128)	The name of the foreign key column.
key_seq	SMALLINT	The key sequence number.
fk_name	CHAR(128)	The foreign key name.
pk_name	CHAR(128)	The primary key name.

Remarks
Foreign keys reference a row in a separate table that contains the corresponding primary key. This procedure allows you to obtain a list of the remote tables with primary keys that correspond to a particular foreign table. The sp_remote_imported_keys result set includes the database, owner, table, column, and name for both the primary and the foreign key, and the foreign key sequence for the foreign key columns. The result set may vary because of the underlying ODBC and JDBC calls, but information about the table and column for a primary key is always returned.

Privileges
None

Side effects
None

Example

The following example returns the tables with primary keys that correspond to a foreign key on the ULOrder table on the remote server named RemoteSA:

```
CALL sp_remote_imported_keys( 'RemoteSA', 'ULOrder', 'DBA' );
```

sp_remote_primary_keys system procedure

Provides primary key information about remote tables using remote data access.

Syntax

```
sp_remote_primary_keys(
 @server_name
 , @table_name
 [, @table_owner
 [, @table_qualifier ] ]
)
```

Arguments

- ***@server_name* –** Use this CHAR(128) parameter to specify the remote server name.
- ***@table_name* –** Use this CHAR(128) parameter to specify the name of the remote table.
- ***@table_owner* –** Use this optional CHAR(128) parameter to specify the owner of the remote table. The default is '%'.
- ***@table_qualifier* –** Use this optional CHAR(128) parameter to specify the name of the remote database. The default is '%'.

Result set

Column name	Data type	Description
database	CHAR(128)	The name of the remote database.
owner	CHAR(128)	The owner of the table.
table_name	CHAR(128)	The name of the table.
column_name	CHAR(128)	The name of the primary key column.
key_seq	SMALLINT	The primary key sequence number.
pk_name	CHAR(128)	The primary key name.

Remarks

This system procedure provides primary key information about remote tables using remote data access.

Because of differences in the underlying ODBC calls, the information returned differs slightly from the catalog/database value depending upon the remote data access class that is specified for the server.

Privileges
None

Standards and compatibility

- **Sybase –** Supported by Open Client/Open Server.

Side effects
None

Examples

The following example returns information about the primary keys in tables owned by DBA in a SAP Sybase IQ remote server named RemoteSA.

```
CALL sp_remote_primary_keys( 'RemoteSA', null, 'DBA' );
```

To get a list of the primary keys in all the tables owned by Fred in the production database in an Adaptive Server Enterprise server named RemoteASE:

```
CALL sp_remote_primary_keys( 'RemoteASE', null, 'Fred',
'production' );
```

sp_remote_tables system procedure

Returns a list of the tables on a server.

Syntax

```
sp_remote_tables(
 @server_name
 [, @table_name
 [, @table_owner
 [, @table_qualifier
 [, @with_table_type ] ] ] ]
)
```

Arguments

- ***@server_name* –** Use this CHAR(128) parameter to specify the remote server name.
- ***@table_name* –** Use this optional CHAR(128) parameter to specify the name of the remote table. The default is '%'.
- ***@table_owner* –** Use this optional CHAR(128) parameter to specify the owner of the remote table. The default is '%'.
- ***@table_qualifier* –** Use this optional CHAR(128) parameter to specify the database in which *table_name* is located. The default is '%'.

- ***@with_table_type* –** Use this optional BIT parameter to specify the inclusion of remote table types. The default is 0. Specify 1 if you want the result set to include a column that lists table types or specify 0 if you do not.

Result set

Column name	Data type	Description
database	CHAR(128)	The name of the remote database.
owner	CHAR(128)	The name of the table owner.
table_name	CHAR(128)	The name of the table.
table_type	CHAR(128)	Specifies the table type. The value depends on the type of remote server. For example, TABLE, VIEW, SYS, and GBL TEMP are possible values.

Remarks
The server must be defined with the CREATE SERVER statement to use this system procedure.

It may be helpful when you are configuring your database server to get a list of the remote tables available on a particular server. This procedure returns a list of the tables on a server.

The procedure accepts five parameters. If a table, owner, or database name is given, the list of tables will be limited to only those that match the arguments.

Privileges
None

Side effects
None

Standards and compatibility

- **Sybase –** Supported by Open Client/Open Server.

Examples

The following example returns information about the tables owned by DBA in a SAP Sybase IQ remote server named RemoteSA.

```
CALL sp_remote_tables( 'RemoteSA', null, 'DBA' );
```

To get a list of all the tables owned by Fred in the production database in an Adaptive Server Enterprise server named RemoteASE:

```
CALL sp_remote_tables( 'RemoteASE', null, 'Fred', 'production' );
```

To get a list of all the Microsoft Excel worksheets available from an ODBC data source referenced by a server named RemoteExcel:

```
CALL sp_remote_tables( 'RemoteExcel' );
```

sp_servercaps system procedure

Displays information about a remote server's capabilities.

Syntax

```
sp_servercaps( @server_name )
```

Arguments

- ***@server_name* –** Use this CHAR(128) parameter to specify a server defined with the CREATE SERVER statement. *@server_name* is the same server name used in the CREATE SERVER statement.

Results

Column	Type	Description
capid	INTEGER	The capability identifier.
capname	CHAR(128)	The name of the capability.
capvalue	CHAR(128)	The setting of the capability, usually T (true) or F (false).

Remarks

The server must be defined with the CREATE SERVER statement to use this system procedure.

This procedure displays information about a remote server's capabilities. The capability information is used to determine how much of a SQL statement can be forwarded to a remote server. The ISYSCAPABILITY system table, which lists the server capabilities, is not populated until a connection is made to the first remote server.

Standards and compatibility

- **Sybase –** Supported by Open Client/Open Server.

Privileges

None

Side effects

None

Example

To display information about the remote server RemoteSA:

```
CALL sp_servercaps( 'RemoteSA' );
```

sp_sys_priv_role_info System Procedure

Generates a report to map a system privilege to the corresponding system role. A single row is returned for each system privilege.

Syntax

sp_sys_priv_role_info()

Result Set

Column Name	Data Type	Description
sys_priv_name	char(128)	The name of the system privi-lege
sys_priv_role_name	char(128)	The role name corresponding to the system privilege.
sys_priv_id	unsigned int	The id of the system privilege.

Privileges

none

sp_tsql_environment system procedure

Sets connection options when users connect from jConnect or Open Client applications.

Syntax

```
sp_tsql_environment( )
```

Remarks

The sp_login_environment procedure is the default procedure specified by the login_procedure database option. For each new connection, the procedure specified by login_procedure is called. If the connection uses the TDS communications protocol (that is, if it is an Open Client or jConnect connection), then sp_login_environment in turn calls sp_tsql_environment.

This procedure sets database options so that they are compatible with default Adaptive Server Enterprise behavior.

To change the default behavior, create new procedures and alter your login_procedure option to point to these new procedures.

Below is the list of the options set by sp_tsql_environment procedure:

```
if db_property( 'IQStore' ) = 'Off' then
    -- SAP Sybase IQ datastore
    SET TEMPORARY OPTION close_on_endtrans='OFF';
end if;
SET TEMPORARY OPTION ansinull='OFF';
SET TEMPORARY OPTION tsql_variables='ON';
SET TEMPORARY OPTION ansi_blanks='ON';
SET TEMPORARY OPTION chained='OFF';
SET TEMPORARY OPTION quoted_identifier='OFF';
SET TEMPORARY OPTION allow_nulls_by_default='OFF';
SET TEMPORARY OPTION on_tsql_error='CONTINUE';
SET TEMPORARY OPTION isolation_level='1';
SET TEMPORARY OPTION date_format='YYYY-MM-DD';
SET TEMPORARY OPTION timestamp_format='YYYY-MM-DD HH:NN:SS.SSS';
SET TEMPORARY OPTION time_format='HH:NN:SS.SSS';
SET TEMPORARY OPTION date_order='MDY';
SET TEMPORARY OPTION escape_character='OFF';
```

Privileges
None

Side effects
None

Example

The example below calls the sp_tsql_environment procedure:

```
CALL sp_tsql_environment();
```

sp_use_secure_feature_key System Procedure

Enables an existing secure feature key.

Syntax

```
sp_use_secure_feature_key (
    name,
    sfkey)
```

Arguments

- **name –** the VARCHAR (128) name of the secure feature key to be enabled.
- **sfkey –** the CHAR (128) authorization key for the secure feature key being enabled. The authorization key must be at least six characters.

Remarks
This procedure enables the secure features that are turned on by the specified secure feature key.

Privileges
None.

Adaptive Server Enterprise System and Catalog Procedures

Adaptive Server Enterprise provides system and catalog procedures to carry out many administrative functions and to obtain system information. SAP Sybase IQ supports some of these procedures.

System procedures are built-in stored procedures used for getting reports from and updating system tables. Catalog stored procedures retrieve information from the system tables in tabular form.

Note: While these procedures perform the same functions as they do in Adaptive Server Enterprise, they are not identical. If you have preexisting scripts that use these procedures, you might want to examine the procedures. To see the text of a stored procedure, run:

```
sp_helptext 'owner.procedure_name'
```

For all system stored procedures delivered by SAP Sybase, the owner is `dbo`. To see the text of a stored procedure of the same name owned by a different user, you must specify that user, for example:

```
sp_helptext 'myname.myprocedure'
```

Adaptive Server Enterprise System Procedures

Adaptive Server Enterprise system procedures provided in SAP Sybase IQ:

Table 149. ASE system procedures provided in SAP Sybase IQ

System procedure	Description	Permissions
sp_addlogin *userid, password[, defdb [, deflanguage [, fullname]]]*	Adds a new user account to a database	Requires the MANAGE ANY USER system privilege.
sp_addmessage *message-num, message_text [, language]*	Adds user-defined messages to `SYSUSERMESSAGES` for use by stored procedure **PRINT** and **RAISERROR** calls	Requires the CREATE MESSAGE or CREATE ANY OBJECT system privilege.
sp_addtype *typename, data-type, [, "identity" | nulltype]*	Creates a user-defined data type. SAP Sybase IQ does not support IDENTITY columns.	Requires the CREATE DATATYPE or CREATE ANY OBJECT system privilege.
sp_adduser *userid [, name_in_db [, grpname]]*	Adds a new user to a database	Requires MANAGE ANY USER system privilege to create a new user. Requires MANAGE ANY USER and MANAGE ROLES system privileges to create a new user and add the user to the role specified.

System procedure	Description	Permissions
sp_dboption *[dbname, optname, {true | false}]*	Displays or changes database options	None required.
sp_droplogin *userid*	Drops a user from a database	Requires MANAGE ANY LOGIN POLICY system privilege.
sp_dropmessage *mes-sage-number [, language]*	Drops user-defined messages	Requires the DROP MESSAGE sys-tem privilege.
sp_droptype *typename*	Drops a user-defined data type	Requires the DROP DATATYPE system privilege.
sp_dropuser *userid*	Drops a user from a database	Requires the MANAGE ANY USER system privilege.
sp_getmessage *message-num, @msg-var output [, language]*	Retrieves stored message strings from `SYSUSERMES-SAGES` for **PRINT** and **RAIS-ERROR** statements.	None required.
sp_helptext '*owner.object-name*'	Displays the text of a system procedure or view	None required.
sp_password *call-er_passwd, new_passwd [, userid]*	Adds or changes a password for a user ID	No system privilege is required to change your own password. The CHANGE PASSWORD system privilege is required to change the password of another user.

Note: Procedures like **sp_dropuser** provide minimal compatibility with Adaptive Server Enterprise stored procedures. If you are accustomed to Adaptive Server Enterprise, compare their text with SAP Sybase IQ procedures before using the procedure in Interactive SQL. To compare, use the command:

```
sp_helptext 'owner.procedure_name'
```

For system stored procedures delivered by Sybase, the owner is always `dbo`. To see the text of a stored procedure of the same name owned by a different user, you must specify that user, for example:

```
sp_helptext 'myname.myprocedure'
```

See also

- *Users, Groups/Roles, and Permissions* on page 829

Adaptive Server Enterprise Catalog Procedures

SAP Sybase IQ implements most of the Adaptive Server Enterprise catalog procedures with the exception of the `sp_column_privileges` procedure.

SAP Sybase IQ also has similar customized stored procedures for some of these Adaptive Server Enterprise catalog procedures.

Table 150. ASE catalog procedures implemented in SAP Sybase IQ

ASE catalog procedure	Description	IQ procedure
sp_columns *table-name [, table-owner] [, table-qualifier] [, column-name]*	Returns the data types of the specified column	
sp_fkeys *pktable_name [, pktable-owner][, pktable-qualifier] [, fktable-name] [, fktable_owner] [, fktable-qualifier]*	Returns foreign-key information about the specified table	
sp_pkeys *table-name [, table_owner] [, table_qualifier]*	Returns primary-key information for a single table	**sp_iqpkeys**
sp_special_columns *table_name [, table-owner] [, table-qualifier] [, col-type]*	Returns the optimal set of columns that uniquely identify a row in a table	
sp_sproc_columns *proc-name [, proc_owner] [, proc-qualifier] [, column-name]*	Returns information about the input and return parameters of a stored procedure	**sp_iqprocparm**
sp_stored_procedures *[sp-name] [, sp-owner] [, sp-qualifier]*	Returns information about one or more stored procedures	**sp_iqprocedure**
sp_tables *table-name [, table-owner] [, table-qualifier] [, table-type]*	Returns a list of objects that can appear in a **FROM** clause	

The following Adaptive Server Enterprise catalog procedures are not supported:

- `sp_column_privileges`
- `sp_databases`
- `sp_datatype_info`
- `sp_server_info`

System Tables and Views

SAP Sybase IQ supports system tables, system views, consolidated views, compatibility views, and Adaptive Server Enterprise T-SQL compatibility views.

System Tables

The structure of every SAP Sybase IQ database is described in a number of system tables. The system tables are designed for internal use.

The `DUMMY`system table is the only system table you are permitted to access directly. For all other system tables, you access their underlying data through their corresponding views.

Table 151. List of system tables

System Table	Internal Use Only?
DUMMY	No
ISYSARTICLE	Yes
ISYSARTICLECOL	Yes
ISYSATTRIBUTE	Yes
ISYSATTRIBUTENAME	Yes
ISYSCAPABILITY	Yes
ISYSCHECK	Yes
ISYSCOLPERM	Yes
ISYSCOLSTAT	Yes
ISYSCONSTRAINT	Yes
ISYSDBFILE	Yes
ISYSDBSPACE	Yes
ISYSDBSPACEPERM	Yes
ISYSDEPENDENCY	Yes
ISYSDOMAIN	Yes
ISYSEVENT	Yes
ISYSEXTERNENV	Yes

System Table	Internal Use Only?
ISYSEXTERNENVOBJECT	Yes
ISYSEXTERNLOGIN	Yes
ISYSFKEY	Yes
ISYSGROUP	Yes
ISYSHISTORY	Yes
ISYSIDX	Yes
ISYSIDXCOL	Yes
ISYSIQBACKUPHISTORY	Yes
ISYSIQBACKUPHISTORYDETAIL	Yes
ISYSIQDBFILE	Yes
ISYSIQDBSPACE	Yes
ISYSIQIDX	Yes
ISYSIQINFO	Yes
ISYSIQLOGICALSERVER	Yes
ISYSIQLOGINPOLICYLSINFO	Yes
ISYSIQLSLOGINPOLICYOPTION	Yes
ISYSIQLSMEMBER	Yes
ISYSIQLSPOLICY	Yes
ISYSIQLSPOLICYOPTION	Yes
ISYSIQMPXSERVER	Yes
ISYSIQPARTITIONCOLUMN	Yes
ISYSIQTAB	Yes
ISYSIQTABCOL	Yes
ISYSJAR	Yes
ISYSJARCOMPONENT	Yes
ISYSJAVACLASS	Yes
ISYSLOGINMAP	Yes
ISYSLOGINPOLICY	Yes

System Table	Internal Use Only?
ISYSLOGINPOLICYOPTION	Yes
ISYSMVOPTION	Yes
ISYSMVOPTIONNAME	Yes
ISYSOBJECT	Yes
ISYSOPTION	Yes
ISYSOPTSTAT	Yes
ISYSPARTITION	Yes
ISYSPARTITIONKEY	Yes
ISYSPARTITIONSCHEME	Yes
ISYSPHYSIDX	Yes
ISYSPROCEDURE	Yes
ISYSPROCPARM	Yes
ISYSPROCPERM	Yes
ISYSPROXYTAB	Yes
ISYSPUBLICATION	Yes
ISYSREMARK	Yes
ISYSREMOTEOPTION	Yes
ISYSREMOTEOPTIONTYPE	Yes
ISYSREMOTETYPE	Yes
ISYSREMOTEUSER	Yes
ISYSSCHEDULE	Yes
ISYSSERVER	Yes
ISYSSOURCE	Yes
ISYSSQLSERVERTYPE	Yes
ISYSSUBPARTITIONKEY	Yes
ISYSSUBSCRIPTION	Yes
ISYSSYNC	Yes
ISYSSYNCPROFILE	Yes

System Table	Internal Use Only?
ISYSSYNCSCRIPT	Yes
ISYSTAB	Yes
ISYSTABCOL	Yes
ISYSTABLEPERM	Yes
ISYSTEXTCONFIG	Yes
ISYSTEXTIDX	Yes
ISYSTEXTIDXTAB	Yes
ISYSTRIGGER	Yes
ISYSTYPEMAP	Yes
ISYSUSER	Yes
ISYSUSERAUTHORITY	Yes
ISYSUSERMESSAGE	Yes
ISYSUSERTYPE	Yes
ISYSVIEW	Yes
ISYSWEBSERVICE	Yes

SYS.DUMMY Table Versus IQ_DUMMY Table

The DUMMY system table is provided as a table that always has exactly one row.

This can be useful for extracting information from the database, as in the following example that gets the current user ID and the current date from the database

```
SELECT USER, today(*) FROM SYS.DUMMY
```

Queries using the `DUMMY` table are run by SQL Anywhere (the catalog store), rather than by SAP Sybase IQ. You can create a dummy table in the SAP Sybase IQ database. For example:

```
CREATE TABLE iq_dummy (dummy_col INT NOT NULL);
```

and use this table explicitly:

```
SELECT NOW() FROM iq_dummy;
```

DUMMY system table

Column name	Column type	Column constraint	Table constraints
dummy_col	INTEGER	NOT NULL	

The DUMMY table is provided as a read-only table that always has exactly one row. This can be useful for extracting information from the database, as in the following example that gets the current user ID and the current date from the database.

```
SELECT USER, today(*) FROM IQ.DUMMY;
```

Use of IQ.DUMMY in the FROM clause is optional. If no table is specified in the FROM clause, the table is assumed to be IQ.DUMMY. The above example could be written as follows:

```
SELECT USER, today(*);
```

dummy_col – This column is not used. It is present because a table cannot be created with no columns.

The cost of reading from the IQ.DUMMY table is less than the cost of reading from a similar user-created table because there is no latch placed on the table page of IQ.DUMMY.

Access plans are not constructed with scans of the IQ.DUMMY table. Instead, references to IQ.DUMMY are replaced with a Row Constructor algorithm, which virtualizes the table reference. This eliminates contention associated with the use of IQ.DUMMY. DUMMY still appears as the table and/or correlation name in short, long, and graphical plans.

System Views

Use the system views to view the contents of the system tables.

A number of predefined system views are provided that present the information in the system tables in a readable format.

The definitions for the system views are included with their descriptions. Some of these definitions are complicated, but you do not need to understand them to use the views.

Consolidated Views

Consolidated views provide data in a form more frequently required by users.

For example, consolidated views often provide commonly needed joins. Consolidated views differ from system views in that they are not just a straightforward view of raw data in an underlying system table. For example, many of the columns in the system views are unintelligible ID values, whereas in the consolidated views, they are readable names.

Compatibility Views

Compatibility views are deprecated views provided for compatibility with earlier versions of SQL Anywhere and SAP Sybase IQ.

Where possible, use system views and consolidated views instead of compatibility views, as support for compatibility views may be eliminated in future versions of SAP Sybase IQ.

Alphabetical List of System Views

System tables are hidden; however, there is a system view for each table. To ensure compatibility with future versions of the IQ main store, make sure your applications use system views and not the underlying system tables, which may change.

ASE T-SQL Compatibility Views

SAP Sybase IQ provides a set of views owned by the special user DBO, which correspond to the Adaptive Server Enterprise system tables and views.

See also

- *Transact-SQL Compatibility Views* on page 809

SYSARTICLE system view

Each row of the SYSARTICLE system view describes an article in a publication. The underlying system table for this view is ISYSARTICLE.

Column name	Data type	Description
publication_id	UNSIGNED INT	The publication of which the article is a part.
table_id	UNSIGNED INT	Each article consists of columns and rows from a single table. This column contains the table ID for this table.
where_expr	LONG VARCHAR	For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.
subscribe_by_expr	LONG VARCHAR	For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.
query	CHAR(1)	Indicates information about the article type to the database server.
alias	VARCHAR(256)	The alias for the article.
schema_change_active	BIT	1 if the table and publication are part of a synchronization schema change.

Constraints on underlying system table

```
PRIMARY KEY (publication_id, table_id)
```

```
FOREIGN KEY (publication_id) REFERENCES SYS.ISYSPUBLICATION
(publication_id)
```

```
FOREIGN KEY (table_id) REFERENCES SYS.ISYSTAB (table_id)
```

See also

- *SYSARTICLECOL system view* on page 695
- *SYSARTICLECOLS consolidated view* on page 695

SYSARTICLECOL system view

Each row of the SYSARTICLECOL system view identifies a column in an article. The underlying system table for this view is ISYSARTICLECOL.

Column name	Data type	Description
publication_id	UNSIGNED INT	A unique identifier for the publication of which the column is a part.
table_id	UNSIGNED INT	The table to which the column belongs.
column_id	UNSIGNED INT	The column identifier, from the SYSTABCOL system view.

Constraints on underlying system table

```
PRIMARY KEY (publication_id, table_id, column_id)
```

```
FOREIGN KEY (publication_id, table_id) REFERENCES SYS.ISYSARTICLE
(publication_id, table_id)
```

```
FOREIGN KEY (table_id, column_id) REFERENCES SYS.ISYSTABCOL
(table_id, column_id)
```

SYSARTICLECOLS consolidated view

Each row in the SYSARTICLECOLS view identifies a column in an article.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSARTICLECOLS"
  as select p.publication_name,t.table_name,c.column_name
    from SYS.ISYSARTICLECOL as ac
      join SYS.ISYSPUBLICATION as p on p.publication_id =
ac.publication_id
      join SYS.ISYSTAB as t on t.table_id = ac.table_id
      join SYS.ISYSTABCOL as c on c.table_id = ac.table_id
      and c.column_id = ac.column_id
```

SYSARTICLES consolidated view

Each row in the SYSARTICLES view describes an article in a publication.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSARTICLES"
  as select u1.user_name as publication_owner,p.publication_name,
    u2.user_name as table_owner,t.table_name,
    a.where_expr,a.subscribe_by_expr,a.alias
    from SYS.ISYSARTICLE as a
      join SYS.ISYSPUBLICATION as p on(a.publication_id =
p.publication_id)
      join SYS.ISYSTAB as t on(a.table_id = t.table_id)
      join SYS.ISYSUSER as u1 on(p.creator = u1.user_id)
      join SYS.ISYSUSER as u2 on(t.creator = u2.user_id)
```

SYSCAPABILITIES consolidated view

Each row in the SYSCAPABILITIES view specifies the status of a capability for a remote database server. This view gets its data from the ISYSCAPABILITY and ISYSCAPABILITYNAME system tables.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCAPABILITIES"
  as select
ISYSCAPABILITY.capid,ISYSCAPABILITY.srvid,property('RemoteCapabilit
y',ISYSCAPABILITY.capid) as capname,ISYSCAPABILITY.capvalue
    from SYS.ISYSCAPABILITY
```

SYSCAPABILITY system view

Each row of the SYSCAPABILITY system view specifies the status of a capability on a remote database server. The underlying system table for this view is ISYSCAPABILITY.

Column name	Data type	Description
capid	INTEGER	The ID of the capability, as listed in the SYSCAPABILITYNAME system view.
srvid	UNSIGNED INT	The server to which the capability applies, as listed in the SYSSERVER system view.
capvalue	CHAR(128)	The value of the capability.

Constraints on underlying system table

```
PRIMARY KEY (capid, srvid)
```

```
FOREIGN KEY (srvid) REFERENCES SYS.ISYSSERVER (srvid)
```

SYSCAPABILITYNAME system view

Each row in the SYSCAPABILITYNAME system view provides a name for each capability ID in the SYSCAPABILITY system view.

Column name	Data type	Description
capid	INTEGER	A number uniquely identifying the capability.
capname	VARCHAR(32000)	The name of the capability.

Remarks

The SYSCAPABILITYNAME system view is defined using a combination of sa_rowgenerator and the following server properties:

RemoteCapability
MaxRemoteCapability

SYSCATALOG consolidated view

Each row in the SYSCATALOG view describes a system table.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCATALOG"( creator,
  tname,dbspacename,tabletype,ncols,primary_key,"check",
  remarks )
  as select u.user_name,tab.table_name,dbs.dbspace_name,
    if tab.table_type_str = 'BASE' then 'TABLE' else
tab.table_type_str endif,
    (select count() from SYS.ISYSTABCOL
      where ISYSTABCOL.table_id = tab.table_id),
    if ix.index_id is null then 'N' else 'Y' endif,
    null,
    rmk.remarks
    from SYS.SYSTAB as tab
     join SYS.ISYSDBSPACE as dbs on(tab.dbspace_id = dbs.dbspace_id)
      join SYS.ISYSUSER as u on u.user_id = tab.creator
     left outer join SYS.ISYSIDX as ix on(tab.table_id = ix.table_id
and ix.index_id = 0)
      left outer join SYS.ISYSREMARK as rmk on(tab.object_id =
rmk.object_id)
```

SYSCERTIFICATE System View

Each row of the SYSCERTIFICATE system view stores a certificate in text PEM-format. The underlying system table for this view is ISYSCERTIFICATE.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The ID of the certificate.
cert_name	CHAR(128)	The certificate name.
contents	LONG BINARY	The certificate contents in a compressed form.
update_time	TIMESTAMP	The local date and time of the last create or replace.
update_time_utc	TIMESTAMP WITH TIME ZONE	The UTC date and time of the last create or replace.

Constraints on Underlying System Table

```
PRIMARY KEY (object_id)
```

```
UNIQUE INDEX (cert_name)
```

SYSCHECK system view

Each row in the SYSCHECK system view provides the definition for a named check constraint in a table. The underlying system table for this view is ISYSCHECK.

Column name	Data type	Description
check_id	UNSIGNED INT	A number that uniquely identifies the constraint in the database.
check_defn	LONG VARCHAR	The CHECK expression.

Constraints on underlying system table

```
PRIMARY KEY (check_id)
```

```
FOREIGN KEY (check_id) REFERENCES SYS.ISYSCONSTRAINT (constraint_id)
```

SYSCOLAUTH consolidated view

Each row in the SYSCOLAUTH view describes the set of privileges (UPDATE, SELECT, or REFERENCES) granted on a column.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW
"SYS"."SYSCOLAUTH"( grantor,grantee,creator,tname,colname,
  privilege_type,is_grantable )
  as select u1.user_name,u2.user_name,u3.user_name,tab.table_name,
    col.column_name,cp.privilege_type,cp.is_grantable
    from SYS.ISYSCOLPERM as cp
      join SYS.ISYSUSER as u1 on u1.user_id = cp.grantor
      join SYS.ISYSUSER as u2 on u2.user_id = cp.grantee
      join SYS.ISYSTAB as tab on tab.table_id = cp.table_id
      join SYS.ISYSUSER as u3 on u3.user_id = tab.creator
      join SYS.ISYSTABCOL as col on col.table_id = cp.table_id
      and col.column_id = cp.column_id
```

SYSCOLLATION compatibility view (deprecated)

The SYSCOLLATION compatibility view contains the collation sequence information for the database. It is obtainable via built-in functions and is not kept in the catalog. Following is definition for this view:

```
ALTER VIEW "SYS"."SYSCOLLATION"
  as select 1 as collation_id,
    DB_PROPERTY('Collation') as collation_label,
    DB_EXTENDED_PROPERTY('Collation','Description') as
collation_name,
    cast(DB_EXTENDED_PROPERTY('Collation','LegacyData') as
binary(1280)) as collation_order
```

SYSCOLLATIONMAPPINGS compatibility view (deprecated)

The SYSCOLLATIONMAPPINGS compatibility view contains only one row with the database collation mapping. It is obtainable via built-in functions and is not kept in the catalog. Following is definition for this view:

```
ALTER VIEW "SYS"."SYSCOLLATIONMAPPINGS"
  as select DB_PROPERTY('Collation') as collation_label,
    DB_EXTENDED_PROPERTY('Collation','Description') as
collation_name,
    DB_PROPERTY('Charset') as cs_label,
    DB_EXTENDED_PROPERTY('Collation','ASESensitiveSortOrder') as
so_case_label,
    DB_EXTENDED_PROPERTY('Collation','ASEInsensitiveSortOrder') as
so_caseless_label,
    DB_EXTENDED_PROPERTY('Charset','java') as jdk_label
```

SYSCOLPERM system view

The GRANT statement can give UPDATE, SELECT, or REFERENCES privileges to individual columns in a table. Each column with UPDATE, SELECT, or REFERENCES privileges is recorded in one row of the SYSCOLPERM system view. The underlying system table for this view is ISYSCOLPERM.

Column name	Data type	Description
table_id	UNSIGNED INT	The table number for the table containing the column.

Column name	Data type	Description
grantee	UNSIGNED INT	The user number of the user ID that is given privilege on the column. If the grantee the PUBLIC role, the privilege is given to all user IDs.
grantor	UNSIGNED INT	The user number of the user ID that grants the privilege.
column_id	UNSIGNED INT	This column number, together with the table_id, identifies the column for which privilege has been granted.
privilege_type	SMALLINT	The number in this column indicates the kind of column privilege (16=REFERENCES, 1=SELECT, or 8=UPDATE).
is_grantable	CHAR(1)	Indicates if the privilege on the column was granted WITH GRANT OPTION.

Constraints on underlying system table

```
PRIMARY KEY (table_id, grantee, grantor, column_id, privilege_type)
```

```
FOREIGN KEY (table_id, column_id) REFERENCES SYS.ISYSTABCOL
(table_id, column_id)
```

```
FOREIGN KEY (grantor) REFERENCES SYS.ISYSUSER (user_id)
```

```
FOREIGN KEY (grantee) REFERENCES SYS.ISYSUSER (user_id)
```

SYSCOLSTAT system view

The SYSCOLSTAT system view contains the column statistics, including histograms, that are used by the optimizer. The contents of this view are best retrieved using the sa_get_histogram stored procedure or the Histogram utility. The underlying system table for this view is ISYSCOLSTAT.

Column name	Data type	Description
table_id	UNSIGNED INT	A number that uniquely identifies the table or materialized view to which the column belongs.
column_id	UNSIGNED INT	A number that, together with table_id, uniquely identifies the column.

Column name	Data type	Description
format_id	SMALLINT	For system use only.
update_time	TIMESTAMP	The local time of the last update of the column statistics.
density	FLOAT	An estimate of the average selectivity of a single value for the column, not counting the large single value selectivities stored in the row.
max_steps	SMALLINT	For system use only.
actual_steps	SMALLINT	For system use only.
step_values	LONG BINARY	For system use only.
frequencies	LONG BINARY	For system use only.
update_time_utc	TIMESTAMP WITH TIME ZONE	The UTC time of the last update of the column statistics.

Constraints on underlying system table

```
PRIMARY KEY (table_id, column_id)
```

```
FOREIGN KEY (table_id, column_id) REFERENCES SYS.ISYSTABCOL
(table_id, column_id)
```

SYSCOLSTATS consolidated view

The SYSCOLSTATS view contains the column statistics that are stored as histograms and used by the optimizer.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCOLSTATS" AS SELECT u.user_name, t.table_name,
c.column_name, s.format_id,
   dateadd(mi, PROPERTY('TimeZoneAdjustment'), s.update_time) as
update_time, s.density, s.max_steps, s.actual_steps,
   s.step_values, s.frequencies , TODATETIMEOFFSET( s.update_time,
0 ) as update_time_utc
FROM SYS.ISYSCOLSTAT s
JOIN SYS.ISYSTABCOL c on (s.table_id = c.table_id and s.column_id =
c.column_id)
JOIN SYS.ISYSTAB t on (t.table_id = c.table_id)
JOIN SYS.ISYSUSER u on (u.user_id = t.creator)
```

SYSCOLUMN compatibility view (deprecated)

The SYSCOLUMN view is provided for compatibility with older versions of the software that offered a SYSCOLUMN system table. However, the previous SYSCOLUMN table has been replaced by the ISYSTABCOL system table, and its corresponding SYSTABCOL system view, which you should use instead.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCOLUMN"
  as select b.table_id,
    b.column_id,
    if c.sequence is null then 'N' else 'Y' endif as pkey,
    b.domain_id,
    b.nulls,
    b.width,
    b.scale,
    b.object_id,
    b.max_identity,
    b.column_name,
    r.remarks,
    b."default",
    b.user_type,
    b.column_type
    from SYS.SYSTABCOL as b
      left outer join SYS.ISYSREMARK as r on(b.object_id =
r.object_id)
      left outer join SYS.ISYSIDXCOL as c on(b.table_id = c.table_id
and b.column_id = c.column_id and c.index_id = 0)
```

SYSCOLUMNS consolidated view

Each row in the SYSCOLUMNS view describes one column of each table and view in the catalog.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW
"SYS"."SYSCOLUMNS"( creator,cname,tname,coltype,nulls,length,
  syslength,in_primary_key,colno,default_value,
  column_kind,remarks )
  as select
u.user_name,col.column_name,tab.table_name,dom.domain_name,
   col.nulls,col.width,col.scale,if ixcol.sequence is null then 'N'
else 'Y' endif,col.column_id,
    col."default",col.column_type,rmk.remarks
    from SYS.SYSTABCOL as col
      left outer join SYS.ISYSIDXCOL as ixcol on(col.table_id =
ixcol.table_id and col.column_id = ixcol.column_id and
ixcol.index_id = 0)
```

```
      join SYS.ISYSTAB as tab on(tab.table_id = col.table_id)
      join SYS.ISYSDOMAIN as dom on(dom.domain_id = col.domain_id)
      join SYS.ISYSUSER as u on u.user_id = tab.creator
      left outer join SYS.ISYSREMARK as rmk on(col.object_id =
rmk.object_id)
```

SYSCOLUMNS ASE Compatibility View

This view is owned by user DBO. syscolumns contains one row for every column in every table and view, and a row for each parameter in a procedure.

See also

- *Tables in Each Adaptive Server Enterprise Database* on page 810
- *SYSCOMMENTS ASE Compatibility View* on page 703
- *SYSINDEXES ASE Compatibility View* on page 726
- *SYSIQOBJECTS ASE Compatibility View* on page 738
- *SYSIQVINDEX ASE Compatibility View* on page 742
- *SYSOBJECTS ASE Compatibility View* on page 750
- *SYSTYPES ASE Compatibility View* on page 797
- *SYSUSERS ASE Compatibility View* on page 805

SYSCOMMENTS ASE Compatibility View

syscomments contains entries for each view, rule, default, trigger, table constraint, partition, procedure, computed column, function-based index key, and other forms of compiled objects.

This view is owned by user DBO.

The text column contains the original definition statements. If the text column is longer than 255 bytes, the entries span rows. Each object can occupy as many as 65,025 rows.

See also

- *Tables in Each Adaptive Server Enterprise Database* on page 810
- *SYSCOLUMNS ASE Compatibility View* on page 703
- *SYSINDEXES ASE Compatibility View* on page 726
- *SYSIQOBJECTS ASE Compatibility View* on page 738
- *SYSIQVINDEX ASE Compatibility View* on page 742
- *SYSOBJECTS ASE Compatibility View* on page 750
- *SYSTYPES ASE Compatibility View* on page 797
- *SYSUSERS ASE Compatibility View* on page 805

SYSCONSTRAINT system view

Each row in the SYSCONSTRAINT system view describes a named constraint in the database. The underlying system table for this view is ISYSCONSTRAINT.

Column name	Data type	Description
constraint_id	UNSIGNED INT	The unique ID for the constraint.
constraint_type	CHAR(1)	The type of constraint: • **C –** column check constraint • **T –** table constraint • **P –** primary key • **F –** foreign key • **U –** unique constraint
ref_object_id	UNSIGNED BIGINT	The object ID of the column, table, or index to which the constraint applies.
table_object_id	UNSIGNED BIGINT	The object ID of the table to which the constraint applies.
constraint_name	CHAR(128)	The name of the constraint.

Constraints on underlying system table

```
PRIMARY KEY (constraint_id)
```

```
FOREIGN KEY (ref_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
```

```
FOREIGN KEY (table_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
```

```
UNIQUE CONSTRAINT (table_object_id, constraint_name)
```

SYSDBFILE system view

Each row in the SYSDBFILE system view describes a dbspace file. The underlying system table for this view is ISYSDBFILE.

Column name	Data type	Description
dbfile_id	SMALLINT	For internal use only.
dbspace_id	SMALLINT	Each dbspace file in a database is assigned a unique number. The system dbspace contains all system objects and has a dbspace_id of 0.

Column name	Data type	Description
dbfile_name	CHAR(128)	The file name for the dbspace.
file_name	LONG VARCHAR	A unique name for the dbspace. It is used in the CREATE TABLE command.
lob_map	LONG VARBIT	For internal use only.

Constraints on underlying system table

```
PRIMARY KEY (dbfile_id)
```

```
FOREIGN KEY (dbspace_id) REFERENCES SYS.ISYSDBSPACE (dbspace_id)
```

```
UNIQUE index (file_name)
```

SYSDBSPACE system view

Each row in the SYSDBSPACE system view describes a dbspace file. The underlying system table for this view is ISYSDBSPACE.

Column name	Data type	Description
dbspace_id	SMALLINT	Unique number identifying the dbspace. The system dbspace contains all system objects and has a dbspace_id of 0.
object_id	UNSIGNED BIGINT	The object ID of the dbspace.
dbspace_name	CHAR(128)	A unique name for the dbspace. It is used in the CREATE TABLE command.
store_type	TINYINT	For internal use only.

Constraints on underlying system table

```
PRIMARY KEY (dbspace_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

SYSDBSPACEPERM system view

Each row in the SYSDBSPACEPERM system view describes a privilege on a dbspace file. The underlying system table for this view is ISYSDBSPACEPERM.

Column name	Data type	Description
dbspace_id	SMALLINT	Unique number identifying the dbspace. The system dbspace contains all system objects and has a dbspace_id of 0.
grantee	UNSIGNED INT	The user ID of the user getting the privilege.
privilege_type	SMALLINT	The privilege that is granted to the grantee. For example, CREATE gives the grantee privilege to create objects on the dbspace.

Constraints on underlying system table

```
FOREIGN KEY (dbspace_id) REFERENCES SYS.ISYSDBSPACE (dbspace_id)
```

```
FOREIGN KEY (grantee) REFERENCES SYS.ISYSUSER (user_id)
```

SYSDEPENDENCY system view

Each row in the SYSDEPENDENCY system view describes a dependency between two database objects. The underlying system table for this view is ISYSDEPENDENCY.

A dependency exists between two database objects when one object references another object in its definition. For example, if the query specification for a view references a table, the view is dependent on the table. The database server tracks dependencies of views on tables, views, materialized views, and columns.

Column name	Data type	Description
ref_object_id	UNSIGNED BIGINT	The object ID of the referenced object.
dep_object_id	UNSIGNED BIGINT	The ID of the referencing object.

Constraints on underlying system table

```
PRIMARY KEY (ref_object_id, dep_object_id)
```

```
FOREIGN KEY (ref_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
```

```
FOREIGN KEY (dep_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
```

SYSDOMAIN system view

The SYSDOMAIN system view records information about built-in data types (also called domains). The contents of this view does not change during normal operation. The underlying system table for this view is ISYSDOMAIN.

Column name	Data type	Description
domain_id	SMALLINT	The unique number assigned to each data type. These numbers cannot be changed.
domain_name	CHAR(128)	The name of the data type normally found in the CREATE TABLE command, such as CHAR or INTEGER.
type_id	SMALLINT	The ODBC data type. This value corresponds to the value for data_type in the Transact-SQL compatibility dbo.SYSTYPES table.
"precision"	SMALLINT	The number of significant digits that can be stored using this data type. The column value is NULL for non-numeric data types.

Constraints on underlying system table

```
PRIMARY KEY (domain_id)
```

SYSEVENT system view

Each row in the SYSEVENT system view describes an event created with CREATE EVENT. The underlying system table for this view is ISYSEVENT.

Column name	Data type	Description
event_id	UNSIGNED INT	The unique number assigned to each event.
object_id	UNSIGNED BIGINT	The internal ID for the event, uniquely identifying it in the database.
creator	UNSIGNED INT	The user number of the owner of the event. The name of the user can be found by looking in the SYSUSER system view.

Column name	Data type	Description
event_name	VARCHAR(128)	The name of the event.
enabled	CHAR(1)	Indicates whether the event is allowed to fire.

location	CHAR(1)	The location where the event is to fire:
		Y = AT ALL clause and FOR PRIMARY clause specified
		E = AT CONSOLIDATED clause and FOR PRIMARY clause specified
		T = AT REMOTE clause and FOR PRIMARY clause specified
		P = (AT clause not specified) FOR PRIMARY clause specified
		B = AT ALL clause and FOR ALL clause specified
		D = AT CONSOLIDATED clause and FOR ALL clause specified
		S = AT REMOTE clause and FOR ALL clause specified
		M = (AT clause not specified) FOR ALL clause specified
		C = AT CONSOLIDATED (FOR clause not specified)
		R = AT REMOTE (FOR clause not specified)
		A = AT ALL clause (FOR clause not specified)

event_type_id	UNSIGNED INT	For system events, the event type as listed in the SYSEVENTTYPE system view.
action	LONG VARCHAR	The event handler definition. An obfuscated value indicates a hidden event.

Column name	Data type	Description
external_action	LONG VARCHAR	For system use only.
condition	LONG VARCHAR	The condition used to control firing of the event handler.
remarks	LONG VARCHAR	Remarks for the event; this column comes from ISYSREMARK.
source	LONG VARCHAR	The original source for the event; this column comes from ISYSSOURCE.

Constraints on underlying system table

```
PRIMARY KEY (event_id)
```

```
FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
UNIQUE INDEX (event_name)
```

SYSEVENTTYPE system view

The SYSEVENTTYPE system view defines the system event types that can be referenced by CREATE EVENT.

Column name	Data type	Description
event_type_id	INT	The unique number assigned to each event type.
name	VARCHAR(32000)	The name of the system event type.
description	VARCHAR(32000)	A description of the system event type.

Remarks

The SYSEVENTTYPE system view is defined using a combination of sa_rowgenerator and the following server properties:

EventTypeName
EventTypeDesc
MaxEventType

SYSEXTERNENV system view

Many external runtime environments are supported, including embedded SQL and ODBC applications written in C/C++, and applications written in Java, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

Each row in the SYSEXTERNENV system view describes the information needed to identify and launch each of the external environments. The underlying system table for this view is ISYSEXTERNENV.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	A unique identifier for the external environment.
name	CHAR(128)	This column identifies the name of the external environment or language. It is one of `java`, `perl`, `php`, `clr`, `c_esql32`, `c_esql64`, `c_odbc32`, or `c_odbc64`.
scope	CHAR(1)	This column is either `C` for CONNECTION or `D` for DATABASE respectively. The scope column identifies if the external environment is launched as one-per-connection or one-per-database. For one-per-connection external environments (like PERL, PHP, C_ESQL32, C_ESQL64, C_ODBC32, and C_ODBC64), there is one instance of the external environment for each connection using the external environment. For a one-per-connection, the external environment terminates when the connection terminates. For one-per-database external environments (like JAVA and CLR), there is one instance of the external environment for each database using the external environment. The one-per-database external environment terminates when the database is stopped.

Column name	Data type	Description
support_result_sets	CHAR(1)	This column identifies those external environments that can return result sets. All external environments can return result sets except PERL and PHP.
location	LONG VARCHAR	This column identifies the location on the database server computer where the executable/binary for the external environment can be found. It includes the executable/binary name. This path can either be fully qualified or relative. If the path is relative, then the executable/binary must be in a location where the database server can find it.
options	LONG VARCHAR	This column identifies any options required on the command line to launch the executable associated with the external environment. You should not modify this column.
user_id	UNSIGNED INT	When the external environment is initially launched, it must make a connection back to the database to set things up for the external environment's usage. By default, this connection is made using the DBA user ID, but if the database administrator prefers to have the external environment use a different user ID with MANAGE ANY EXTERNAL OBJECT system privilege, then the user_id column would indicate that different user ID instead. Typically, this column is NULL and the database server, by default, uses the DBA user ID.

Constraints on underlying system table

```
PRIMARY KEY (object_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)
```

```
UNIQUE INDEX (name)
```

SYSEXTERNENVOBJECT system view

Many external runtime environments are supported, including embedded SQL and ODBC applications written in C/C++, and applications written in Java, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

Each row in the SYSEXTERNENVOBJECT system view describes an installed external object. The underlying system table for this view is ISYSEXTERNENVOBJECT.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	A unique identifier for the external object.
extenv_id	UNSIGNED BIGINT	The unique identifier for the external environment (SYSEXTERNENV.object_id).
owner	UNSIGNED INT	This column identifies the creator/owner of the external object.
name	LONG VARCHAR	This column identifies the name of the external object as specified in the INSTALL EXTERNAL OBJECT statement.
contents	LONG BINARY	The contents of the external object.
update_time	TIMESTAMP	This column identifies the last local time the object was modified (or installed).
update_time_utc	TIMESTAMP WITH TIME ZONE	This column identifies the last UTC time the object was modified (or installed).

Constraints on underlying system table

```
PRIMARY KEY (object_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
FOREIGN KEY (extenv_id) REFERENCES SYS.ISYSEXTERNENV (object_id)
```

```
FOREIGN KEY (owner) REFERENCES SYS.ISYSUSER (user_id)
```

```
UNIQUE INDEX (name)
```

SYSEXTERNLOGIN system view

Each row in the SYSEXTERNLOGIN system view describes an external login for remote data access. The underlying system table for this view is ISYSEXTERNLOGIN.

Note: Previous versions of the catalog contained a SYSEXTERNLOGINS system table. That table has been renamed to be ISYSEXTERNLOGIN (without an 'S'), and is the underlying table for this view.

Column name	Data type	Description
user_id	UNSIGNED INT	The user ID on the local database.
srvid	UNSIGNED INT	The remote server, as listed in the SYSSERVER system view.
remote_login	VARCHAR(128)	The login name for the user, for the remote server.
remote_password	VARBINARY(128)	The password for the user, for the remote server.

Constraints on underlying system table

```
PRIMARY KEY (user_id, srvid)
```

```
FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)
```

```
FOREIGN KEY (srvid) REFERENCES SYS.ISYSSERVER (srvid)
```

SYSFILE compatibility view (deprecated)

Each row in the SYSFILE system view describes a dbspace for a database. Every database consists of one or more dbspaces; each dbspace corresponds to an operating system file.

dbspaces are automatically created for the main database file, temporary file, transaction log file, and transaction log mirror file. Information about the transaction log, and transaction log mirror dbspaces does not appear in the SYSFILE system view.

```
ALTER VIEW "SYS"."SYSFILE"
  as select b.dbfile_id as file_id,
    if b.dbspace_id = 0 and b.dbfile_id = 0 then
      db_property('File')
    else
      if b.dbspace_id = 15 and b.dbfile_id = 15 then
        db_property('TempFileName')
      else
        b.file_name
      endif
    endif as file_name,
    a.dbspace_name,
    a.store_type,
    b.lob_map,
    b.dbspace_id
    from SYS.ISYSDBSPACE as a
      join SYS.ISYSDBFILE as b on(a.dbspace_id = b.dbspace_id)
```

SYSFKCOL compatibility view (deprecated)

Each row of SYSFKCOL describes the association between a foreign column in the foreign table of a relationship and the primary column in the primary table. This view is deprecated; use the SYSIDX and SYSIDXCOL system views instead.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSFKCOL"
  as select a.table_id as foreign_table_id,
    a.index_id as foreign_key_id,
    a.column_id as foreign_column_id,
    a.primary_column_id
    from SYS.ISYSIDXCOL as a
      ,SYS.ISYSIDX as b
    where a.table_id = b.table_id
    and a.index_id = b.index_id
    and b.index_category = 2
```

SYSFKEY system view

Each row in the SYSFKEY system view describes a foreign key constraint in the system. The underlying system table for this view is ISYSFKEY.

Column name	Data type	Description
foreign_table_id	UNSIGNED INT	The table number of the foreign table.
foreign_index_id	UNSIGNED INT	The index number for the foreign key.
primary_table_id	UNSIGNED INT	The table number of the primary table.
primary_index_id	UNSIGNED INT	The index number of the primary key.
match_type	TINYINT	The matching type for the constraint. Matching types include: • **0 –** Use the default matching • **1 –** SIMPLE • **2 –** FULL • **129 –** SIMPLE UNIQUE • **130 –** FULL UNIQUE

Column name	Data type	Description
check_on_commit	CHAR(1)	Indicates whether INSERT and UPDATE statements should wait until the COMMIT to check if foreign keys are still valid.
nulls	CHAR(1)	Indicates whether the columns in the foreign key are allowed to contain the NULL value. This setting is independent of the nulls setting in the columns contained in the foreign key.

Constraints on underlying system table

```
PRIMARY KEY (foreign_table_id, foreign_index_id)
```

```
FOREIGN KEY (foreign_table_id, foreign_index_id) REFERENCES
SYS.ISYSIDX (table_id, index_id)
```

```
FOREIGN KEY (primary_table_id, primary_index_id) REFERENCES
SYS.ISYSIDX (table_id, index_id)
```

SYSFOREIGNKEY compatibility view (deprecated)

The SYSFOREIGNKEY view is provided for compatibility with older versions of the software that offered a SYSFOREIGNKEY system table. However, the previous SYSFOREIGNKEY system table has been replaced by the ISYSFKEY system table, and its corresponding SYSFKEY system view, which you should use instead.

A foreign key is a relationship between two tables—the foreign table and the primary table. Every foreign key is defined by one row in SYSFOREIGNKEY and one or more rows in SYSFKCOL. SYSFOREIGNKEY contains general information about the foreign key while SYSFKCOL identifies the columns in the foreign key and associates each column in the foreign key with a column in the primary key of the primary table.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSFOREIGNKEY"
  as select b.foreign_table_id,
    b.foreign_index_id as foreign_key_id,
    a.object_id,
    b.primary_table_id,
    p.root,
    b.check_on_commit,
    b.nulls,
    a.index_name as role,
    r.remarks,
    b.primary_index_id,
    a.not_enforced as fk_not_enforced,
    10 as hash_limit
    from(SYS.ISYSIDX as a left outer join SYS.ISYSPHYSIDX as p
on(a.table_id = p.table_id and a.phys_index_id = p.phys_index_id))
      left outer join SYS.ISYSREMARK as r on(a.object_id =
r.object_id)
      ,SYS.ISYSFKEY as b
    where a.table_id = b.foreign_table_id
    and a.index_id = b.foreign_index_id
```

SYSFOREIGNKEYS consolidated view

Each row in the SYSFOREIGNKEYS view describes one foreign key for each table in the catalog.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSFOREIGNKEYS"( foreign_creator,
  foreign_tname,
  primary_creator,primary_tname,role,columns )
  as select fk_up.user_name,fk_tab.table_name,pk_up.user_name,
    pk_tab.table_name,ix.index_name,
    (select list(string(fk_col.column_name,' IS ',
      pk_col.column_name)
      order by fkc.table_id,fkc.index_id,fkc."sequence")
      from SYS.ISYSIDXCOL as fkc
        join SYS.ISYSTABCOL as fk_col on(
        fkc.table_id = fk_col.table_id
        and fkc.column_id = fk_col.column_id)
        ,SYS.ISYSTABCOL as pk_col
      where fkc.table_id = fk.foreign_table_id
      and fkc.index_id = fk.foreign_index_id
      and pk_col.table_id = fk.primary_table_id
      and pk_col.column_id = fkc.primary_column_id)
    from SYS.ISYSFKEY as fk
      join SYS.ISYSTAB as fk_tab on fk_tab.table_id =
fk.foreign_table_id
      join SYS.ISYSUSER as fk_up on fk_up.user_id = fk_tab.creator
      join SYS.ISYSTAB as pk_tab on pk_tab.table_id =
fk.primary_table_id
      join SYS.ISYSUSER as pk_up on pk_up.user_id = pk_tab.creator
     join SYS.ISYSIDX as ix on ix.table_id = fk.foreign_table_id and
ix.index_id = fk.foreign_index_id
```

SYSGROUP compatibility view

There is one row in the SYSGROUP system view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups.

Column name	Data type	Description
group_id	UNSIGNED INT	The user number of the group.

Column name	Data type	Description
group_member	UNSIGNED INT	The user number of a member.

Constraints on underlying system table

```
PRIMARY KEY (group_id, group_member)
```

```
FOREIGN KEY group_id (group_id) REFERENCES SYS.ISYSUSER (user_id)
```

```
FOREIGN KEY group_member (group_member) REFERENCES SYS.ISYSUSER
(user_id)
```

SYSGROUPS compatibility view

There is one row in the SYSGROUPS view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSGROUPS"( group_name,
  member_name )
  as select g.user_name,u.user_name
    from SYS.ISYSROLEGRANT,SYS.ISYSUSER as g,SYS.ISYSUSER as u
   where ISYSROLEGRANT.role_id = g.user_id and ISYSROLEGRANT.grantee
= u.user_id and(
    u.user_name in( 'SYS_SPATIAL_ADMIN_ROLE' )
    or u.user_id <= 2147483648) and(
    g.user_type = (0x02|0x04|0x08)
    or g.user_name in( 'SYS','PUBLIC','dbo','diagnostics',
    'rs_systabgroup','SA_DEBUG','SYS_SPATIAL_ADMIN_ROLE' ) )
```

SYSHISTORY system view

Each row in the SYSHISTORY system view records a system operation on the database, such as a database start, a database calibration, and so on. The underlying system table for this view is ISYSHISTORY.

Column name	Data type	Description
operation	CHAR(128)	The type of operation performed on the database file. The operation must be one of the following values: • **INIT –** Information about when the database was created. • **UPGRADE –** Information about when the database was upgraded. • **START –** Information about when the database was started using a specific version of the database server on a particular operating system. • **LAST_START –** Information about the most recent time the database server was started. A LAST_START operation is converted to a START operation when the database is started with a different version of the database server and/or on a different operating system than those values currently stored in the LAST_START row. • **DTT –** Information about the *second to last* Disk Transfer Time (DTT) calibration operation performed on the dbspace. That is, information about the second to last execution of either an ALTER DATABASE CALIBRATE or ALTER DATABASE RESTORE DEFAULT CALIBRATION statement. • **LAST_DTT –** Information about the *most recent* DTT calibration operation performed on the dbspace. That

Column name	Data type	Description
		is, information about the most recent execution of either an ALTER DATABASE CALIBRATE or ALTER DATABASE RESTORE DEFAULT CALIBRATION statement. • **LAST_BACKUP –** Information about the last backup, including date and time of the backup, the backup type, the files that were backed up, and the version of database server that performed the backup.
object_id	UNSIGNED INT	For any operation other than DTT and LAST_DTT, the value in this column will be 0. For DTT and LAST_DTT operations, this is the dbspace_id of the dbspace as defined in the SYSDBSPACE system view.
sub_operation	CHAR(128)	For any operation other than DTT and LAST_DTT, the value in this column will be a set of empty single quotes (''). For DTT and LAST_DTT operations, this column contains the type of sub-operation performed on the dbspace. Values include: • **DTT_SET –** The dbspace calibration has been set. • **DTT_UNSET –** The dbspace calibration has been restored to the default setting.
version	CHAR(128)	The version and build number of the database server used to perform the operation.
platform	CHAR(128)	The operating system on which the operation was carried out.

Column name	Data type	Description
first_time	TIMESTAMP	The local date and time the database was first started on a particular operating system with a particular version of the software.
last_time	TIMESTAMP	The most recent local date and time the database was started on a particular operating system with a particular version of the software.
details	LONG VARCHAR	This column stores information such as command line options used to start the database server or the capability bits enabled for the database. This information is for use by Technical Support.
first_time_utc	TIMESTAMP WITH TIME ZONE	The UTC date and time the database was first started on a particular operating system with a particular version of the software.
last_time_utc	TIMESTAMP WITH TIME ZONE	The most recent UTC date and time the database was started on a particular operating system with a particular version of the software.

Constraints on underlying system table

```
PRIMARY KEY (operation, object_id, version, platform)
```

SYSIDX system view

Each row in the SYSIDX system view defines a logical index in the database. The underlying system table for this view is ISYSIDX.

Column name	Data type	Description
table_id	UNSIGNED INT	Uniquely identifies the table to which this index applies.
index_id	UNSIGNED INT	A unique number identifying the index within its table.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The internal ID for the index, uniquely identifying it in the database.
phys_index_id	UNSIGNED INT	Identifies the underlying physical index used to implement the logical index. This value is NULL for indexes on temporary tables or remote tables. Otherwise, the value corresponds to the object_id of a physical index in the SYSPHYSIDX system view.
dbspace_id	SMALLINT	The ID of the file in which the index is contained. This value corresponds to an entry in the SYSDBSPACE system view.
index_category	TINYINT	The type of index. Values include: • **1 –** Primary key • **2 –** Foreign key • **3 –** Secondary index (includes unique constraints) • **4 –** Text indexes
"unique"	TINYINT	Indicates whether the index is a unique index (1), a unique constraint (2), reserved (3), a non-unique index (4), or a unique index WITH NULLS NOT DISTINCT. A unique index prevents two rows in the indexed table from having the same values in the index columns.
index_name	CHAR(128)	The name of the index.
not_enforced	CHAR(1)	For system use only.

Column name	Data type	Description
file_id	SMALLINT	DEPRECATED. This column is present in SYSVIEW, but not in the underlying system table ISYSIDX. The contents of this column are the same as dbspace_id and it is provided for compatibility. Use dbspace_id instead.

Constraints on underlying system table

```
PRIMARY KEY (table_id, index_id)
```

```
FOREIGN KEY (table_id) REFERENCES SYS.ISYSTAB (table_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
FOREIGN KEY (table_id, phys_index_id) REFERENCES SYS.ISYSPHYSIDX
(table_id, phys_index_id)
```

```
UNIQUE INDEX (index_name, table_id, index_category)
```

SYSIDXCOL system view

Each row in the SYSIDXCOL system view describes one column of an index described in the SYSIDX system view. The underlying system table for this view is ISYSIDXCOL.

Column name	Data type	Description
table_id	UNSIGNED INT	Identifies the table to which the index applies.
index_id	UNSIGNED INT	Identifies the index to which the column applies. Together, table_id and index_id identify one index described in the SYSIDX system view.
sequence	SMALLINT	Each column in an index is assigned a unique number starting at 0. The order of these numbers determines the relative significance of the columns in the index. The most important column has sequence number 0.

Column name	Data type	Description
column_id	UNSIGNED INT	Identifies which column of the table is indexed. Together, table_id and column_id identify one column described in the SYSCOLUMN system view.
"order"	CHAR(1)	Indicates whether the column in the index is kept in ascending(A) or descending(D) order. This value is NULL for text indexes.
primary_column_id	UNSIGNED INT	The ID of the primary key column that corresponds to this foreign key column. The value is NULL for non foreign key columns.

Constraints on underlying system table

```
PRIMARY KEY (table_id, index_id, column_id)
```

```
FOREIGN KEY (table_id, index_id) REFERENCES SYS.ISYSIDX (table_id,
index_id)
```

```
FOREIGN KEY (table_id, column_id) REFERENCES SYS.ISYSTABCOL
(table_id, column_id)
```

SYSINDEX compatibility view (deprecated)

The SYSINDEX view is provided for compatibility with older versions of the software that offered a SYSINDEX system table. However, the SYSINDEX system table has been replaced by the ISYSIDX system table, and its corresponding SYSIDX system view, which you should use instead.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSINDEX"
  as select b.table_id,
    b.index_id,
    b.object_id,
    p.root,
    b.dbspace_id,
    case b."unique"
    when 1 then 'Y'
    when 2 then 'U'
    when 3 then 'M'
    when 4 then 'N'
    when 5 then 'Y'
    else 'I'
    end as "unique",
    t.creator,
    b.index_name,
    r.remarks,
    10 as hash_limit,
    b.dbspace_id as file_id
    from(SYS.ISYSIDX as b left outer join SYS.ISYSPHYSIDX as p
on(b.table_id = p.table_id and b.phys_index_id = p.phys_index_id))
      left outer join SYS.ISYSREMARK as r on(b.object_id =
r.object_id)
      ,SYS.ISYSTAB as t
    where t.table_id = b.table_id
    and b.index_category = 3
```

SYSINDEXES consolidated view

Each row in the SYSINDEXES view describes one index in the database. As an alternative to this view, you could also use the SYSIDX and SYSIDXCOL system views.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSINDEXES"( icreator,
  iname,fname,creator,tname,indextype,
  colnames,interval,level_num )
  as select u.user_name,idx.index_name,dbs.dbspace_name,u.user_name,
    tab.table_name,
    case idx.index_category
    when 1 then 'Primary Key'
    when 2 then 'Foreign Key'
    when 3 then(
      if idx."unique" = 4 then 'Non-unique'
      else if idx."unique" = 2 then 'UNIQUE constraint'
        else if idx."unique" = 5 then 'UNIQUE NULLS NOT DISTINCT'
          else 'UNIQUE'
          endif
        endif
      endif) when 4 then 'Text Index' end,(select
list(string(c.column_name,
      if ixc."order" = 'A' then ' ASC' else ' DESC' endif) order by
      ixc.table_id asc,ixc.index_id asc,ixc.sequence asc)
      from SYS.ISYSIDXCOL as ixc
        join SYS.ISYSTABCOL as c on(
        c.table_id = ixc.table_id
        and c.column_id = ixc.column_id)
      where ixc.index_id = idx.index_id
      and ixc.table_id = idx.table_id),
    0,0
    from SYS.ISYSTAB as tab
     join SYS.ISYSDBSPACE as dbs on(tab.dbspace_id = dbs.dbspace_id)
      join SYS.ISYSIDX as idx on(idx.table_id = tab.table_id)
      join SYS.ISYSUSER as u on u.user_id = tab.creator
```

SYSINDEXES ASE Compatibility View

sysindexes contains one row for each clustered index, one row for each nonclustered index, one row for each table that has no clustered index, and one row for each table that contains text or image columns.

This table also contains one row for each function-based index or index created on a computed column.

This view is owned by user DBO.

See also

- *Tables in Each Adaptive Server Enterprise Database* on page 810
- *SYSCOLUMNS ASE Compatibility View* on page 703
- *SYSCOMMENTS ASE Compatibility View* on page 703
- *SYSIQOBJECTS ASE Compatibility View* on page 738
- *SYSIQVINDEX ASE Compatibility View* on page 742
- *SYSOBJECTS ASE Compatibility View* on page 750
- *SYSTYPES ASE Compatibility View* on page 797
- *SYSUSERS ASE Compatibility View* on page 805

SYSINFO compatibility view (deprecated)

The SYSINFO view indicates the database characteristics, as defined when the database was created. It always contains only one row. This view is obtainable via built-in functions and is not kept in the catalog. Following is the definition for the SYSINFO view:

```
ALTER VIEW "SYS"."SYSINFO"( page_size,
  encryption,
  blank_padding,
  case_sensitivity,
  default_collation,
  database_version )
  as select db_property('PageSize'),
    if db_property('Encryption') <> 'None' then 'Y' else 'N' endif,
    if db_property('BlankPadding') = 'On' then 'Y' else 'N' endif,
    if db_property('CaseSensitive') = 'On' then 'Y' else 'N' endif,
    db_property('Collation'),
    NULL
```

SYSIQBACKUPHISTORY System View

This view presents group information from `ISYSIQBACKUPHISTORY` in a readable format. Each row in this view describes a particular backup operation that finished successfully.

The view SYSIQBACKUP projects equivalent string values for columns type, subtype, and bkp_virtual.

Column name	Column type	Column constraint	Description
bu_id	unsigned bigint	NOT NULL	Transaction identifier of the checkpoint of the operation. Backup ID for backup operations.
bu_time	timestamp	NOT NULL	Time of backup operation that is recorded in backup record.
type	tinyint	NOT NULL	Backup type: 0 = FULL 1 = INCREMENTAL 2 = INCREMENTAL SINCE FULL
selective_type	tinyint	NOT NULL	Backup subtype: 0 = ALL (backs up all dbfiles) 1 = READ/WRITE ONLY (backs up all read-write files) 2 = READ ONLY (backs up a particular read-only file)
virtual_type	tinyint	NOT NULL	Backup virtual type: 0 = NONE 1 = DECOUPLED 2 = ENCAPSULATED
dependson_id	unsigned bigint	NULL	NULL for FULL backup
cmd	long varchar	NOT NULL	Full text of command
creator	char(128)	NOT NULL	User who issued backup command
version	unsigned int	NOT NULL	Backup version

Constraints on underlying system table:

Primary key (bu_id)

See also

- *sp_iqbackupdetails Procedure* on page 431

SYSIQBACKUPHISTORYDETAIL System View

This view describes all the dbfile records present in the database at backup time. Each row in this view describes a particular backup operation that finished successfully.

It presents group information from `ISYSIQBACKUPHISTORYDETAIL` in a readable format. The column constraint for each column is NOT NULL.

Column name	Column type	Description
bu_id	unsigned bigint	Transaction identifier of the check-point of the operation. Backup ID for backup operation.
dbspace_id	smallint	The dbspace ID of which this dbfile record is associated.
dbfile_id	smallint	The dbfile ID present in dbspace during ongoing backup operation
dbspace_rwstatus	char(1)	T indicates read-write
dbspace_createid	unsigned bigint	The transaction ID of the transac-tion that created the dbspace
dbspace_alterid	unsigned bigint	Transaction ID that marked the dbspace RO. If not marked, then the create ID
dbspace_online	char(1)	T indicates online
dbfile_rwstatus	char(1)	T indicates read-write
dbfile_createid	unsigned bigint	The transaction ID of the transac-tion that created this dbfile
dbfile_alterid	unsigned bigint	The transaction ID of the transac-tion that last altered the read-write status of this dbfile
is_backed_up	char(1)	Indicates that the dbfile is backed up in this backup
start_block	unsigned bigint	Start block for the dbfile
num_blocks	unsigned bigint	Total number of blocks in dbfile
num_blocks_backed_up	unsigned bigint	Total number of blocks backed up
dbspace_name	char(128)	Dbspace name
dbfile_name	char(128)	Logical file name of the dbfile
dbfile_path	long varchar	Physical path of the file

Constraints on underlying system table:

Primary key (bu_id, dbfile_id)

Foreign key (txn_id) references SYS.ISYSBACKUPHISTORY

SYSIQCOLUMN System View (Deprecated)

SYSIQCOLUMN has been replaced by the SYSIQTABCOL system view.

See also

- *SYSIQTABCOL System View* on page 741

SYSIQDBFILE System View

Presents group information from `ISYSIQDBFILE` in a readable format.

Note: This view replaces the deprecated system view SYSIQFILE.

Column name	Column type	Description
dbfile_id	small int	Unique ID for the dbfile
start_block	rowid	Number of the first block
block_count	rowid	Number of blocks for this file (dbspace)
reserve_size	rowid	Pre-allocated file system space for the dbspace
allocated	char(1)	Defines whether the segment is pre-allocated (T) or auto-allocated (F)
data_offset	unsigned int	Identifies the byte location of where the SAP Sybase IQ data starts, relative to the beginning of the raw partition
create_time	timestamp	Date and time the file was created
last_modified	timestamp	Date and time the file was last modified
read_write	char(1)	T indicates read-write
online	char(1)	T indicates online
create_txn_id	unsigned bigint	Transaction ID that created the dbfile

Column name	Column type	Description
alter_txn_id	unsigned bigint	Transaction ID that last modified read_write status
server_id	unsigned int	Multiplex server name
file_name	text	Name of the dbspace, used by multiplex server to open it

Constraints on underlying system table:

Foreign key (server_id) references SYS.ISYSIQMPXSERVER

Unique (server_id, file_name)

See also

- *SYSIQFILE System View (Deprecated)* on page 731

SYSIQDBSPACE System View

Presents group information from `ISYSIQDBSPACE` in a readable format.

Column name	Column type	Description
dbspace_id	small int	Each dbspace in a database is assigned a unique number (dbspace ID)
last_modified	timestamp	Time at which the dbspace's read-write status was last modified
segment_type	char(8)	Segment type: Main, Temp or Msg
read_write	char(1)	'T' – read writable; 'F' – read only
online	char(1)	'T' – online; 'F' – offline
create_txn_id	unsigned bigint	Transaction ID that create the dbspace
alter_txn_id	unsigned bigint	Transaction ID that last modified read_write status
striping_on	char(1)	'T' – disk striping on; 'F' – disk striping off
stripe_size_kb	unsigned int	Number of kilobytes written to each file of the dbspace before the disk striping algorithm moves to the next dbfile

Column name	Column type	Description
is_rlv_store	char(1)	'T' – dbspace is a RLV store dbspace; 'F' – dbspace is a MIN, SHARED_TEMP, or TEMPORARY store dbspace.

Constraints on underlying system table:

Primary key (dbspace_id)

Foreign key (dbspace_id) references SYS.ISYSDBSPACE(dbspace_id)

SYSIQFILE System View (Deprecated)

SYSIQFILE has been replaced by the SYSIQDBFILE system view.

See also

- *SYSIQDBFILE System View* on page 729

SYSIQIDX System View

Presents group information from `ISYSIQIDX` in a readable format. Each row in the `SYSIQIDX` view describes an IQ index.

Note: This view replaces the deprecated system view SYSIQINDEX.

Column name	Column type	Description
table_id	unsigned int	The table number uniquely identifies the table to which this index applies
index_id	unsigned int	Each index for one particular table is assigned a unique index number
index_type	char(4)	Index type
index_owner	char(4)	Index owner
max_key	unsigned int	For internal use
identity_location	hs_vdorecid	For internal use
identity_size	unsigned int	For internal use
identity_location_size	unsigned int	For internal use
link_index_id	unsigned int	For internal use

Column name	Column type	Description
delimited_by	varchar(1024)	(WD indexes only) List of separators used to parse a column's string into the words to be stored in that column's WD index
limit	unsigned int	(WD indexes only) Maximum word length for WD index

Constraints on underlying system table:

Primary key (table_id, index_id)

Foreign key (table_id, index_id) references SYS.ISYIDX

Foreign key (link_table_id, link_index_id, table_id, index_id) references SYS.ISYSIDX

SYSIQINFO System View

Presents group information from `ISYSIQINFO` in a readable format.

The ISYSIQINFO system table indicates the database characteristics as defined when the SAP Sybase IQ database was created using **CREATE DATABASE**. It always contains only one row.

Column name	Column type	Description
create_time	TIMESTAMP NOT NULL	Date and time that the database was created
update_time	TIMESTAMP NOT NULL	Date and time of the last update
file_format_version	UNSIGNED INT NOT NULL	File format number of files for this database
cat_format_version	UNSIGNED INT NOT NULL	Catalog format number for this database
sp_format_version	UNSIGNED INT NOT NULL	Stored procedure format number for this database
block_size	UNSIGNED INT NOT NULL	Block size specified for the database
chunk_size	UNSIGNED INT NOT NULL	Number of blocks per page as determined by the block size and page size specified for the database
file_format_date	CHAR(10) NOT NULL	Date when file format number was last changed
dbsig	BINARY(136) NOT NULL	Used internally by catalog

Column name	Column type	Description
commit_txn_id	unsigned bigint	For internal use
rd_commit_txn_id	unsigned bigint	For internal use
multiplex name	CHAR(128) NULL	Name of the multiplex that this database is a member of
last_multiplex_mode	TINYINT NULL	(Column unused in SAP Sybase IQ 16.0) Mode of the server that last opened the catalog read-write. One of the following values. • 0 – Single Node. • 1 – Reader. • 2 – Coordinator. • 3 – Writer. .

SYSIQLOGICALSERVER System View

Presents a readable version of the ISYSIQLOGICALSERVER system table.

The ISYSIQLOGICALSERVER system table stores logical server information and associated logical server policy information.

Column name	Column type	Description
ls_id	UNSIGNED BIGINT NOT NULL	The ID number of the logical server.
ls_object_id	UNSIGNED BIGINT NOT NULL	The logical server object ID number.
ls_policy_id	UNSIGNED BIGINT NOT NULL	The ID number of the logical server policy.
ls_name	CHAR(128) NOT NULL UNIQUE	The logical server name.

Constraints on underlying system table:

- Primary key(ls_id)
- object_id foreign key(ISYSOBJECT)
- ls_policy_id foreign key(ISYSIQLSPOLICY)

SYSIQLOGINPOLICYLSINFO System View

Presents a readable version of the table ISYSIQLOGINPOLICYLSINFO.

The ISYSIQLOGINPOLICYLSINFO system table stores the login policy logical server assignment information.

Column name	Column type	Description
login_policy_id	UNSIGNED BIGINT NOT NULL	The ID number of the login policy.
ls_id	UNSIGNED BIGINT NOT NULL	The ID number of the logical serv-er.

Constraints on underlying system table:

- Primary key(login_policy_id, ls_id)
- login_policy_id foreign key(ISYSLOGINPOLICY)
- ls_id foreign key(ISYSIQLOGICALSERVER)

SYSIQLSLOGINPOLICIES Consolidated View

Describes all the logical server assignments from the login policies.

This consolidated system view shows information from SYSIQLOGICALSERVER, ISYSIQLOGINPOLICYLSINFO and ISYSLOGINPOLICY.

Column name	Column type	Description
ls_id	UNSIGNED BIGINT NOT NULL	Logical server identifier.
ls_name	CHAR(128)	Logical server name.
login_policy_id	UNSIGNED BIGINT NOT NULL	The ID number of the login policy.
login_policy_name	char(128)	The name of the login policy.

SYSIQLSLOGINPOLICYOPTION System View

Presents a version of the table ISYSIQLSLOGINPOLICYOPTION in a readable format.

The ISYSIQLSLOGINPOLICYOPTION table stores the logical server level settings for login policy option values.

Column name	Column type	Description
login_policy_id	UNSIGNED BIGINT NOT NULL	The ID number of the login policy.

Column name	Column type	Description
ls_id	UNSIGNED BIGINT NOT NULL	Logical server identifier.
login_option_name	CHAR(128) NOT NULL	The name of the login policy option.
login_option_value	LONG VARCHAR NOT NULL	The value of the login policy option.

Constraints on underlying system table:

- Primary key(login_policy_id,ls_id, login_option_name)
- login_policy_id foreign key(ISYSLOGINPOLICY)
- ls_id foreign key(ISYSIQLOGICALSERVER)

SYSIQLSMEMBER System View

Presents group information from the ISYSIQLSMEMBER table, which stores logical server membership information.

`ISYSIQLSMEMBER` stores the logical servers and their corresponding multiplex servers.

Column name	Column type	Description
ls_id	UNSIGNED BIGINT NOT NULL	The ID number of the logical server.
logical_membership_type	TINYNT NOT NULL	The type of the logical membership.
mpx_server_id	UNSIGNED INT NOT NULL	The ID number of the multiplex server.
membership_info	UNSIGNED INT NOT NULL	The membership information.

Constraints on underlying system table:

- Primary key(ls_id, logical_membership_id, mpx_server_id)
- ls_id foreign key(ISYSIQLOGICALSERVER)

For logical server memberships that are defined using the multiplex server name, the value of logical_membership_type is 0 and mpx_server_id is the server id of the multiplex server.

For the logical membership of the coordinator, mpx_server_id is 0 and logical_membership_type is 1.

SYSIQLSMEMBERS Consolidated View

Describes all user-defined logical server memberships.

Column name	Column type	Description
ls_id	UNSIGNED BIGINT NOT NULL	The ID number of the logical server.
ls_name	CHAR(128) NOT NULL	The name of the logical server.
server_id	UNSIGNED INT NOT NULL	The multiplex server identifier of the member, for the membership defined using server name, or 0, for the logical membership of the coordinator.
server_name	CHAR(128) NOT NULL	The multiplex server name of the member for the membership defined using the server name, or 'LOGICAL COORDINATOR' for the logical membership of the coordinator.
membership_type	TINYINT NOT NULL	0 for the membership defined using the server name, or 1 for the logical membership of the coordinator.

SYSIQLSPOLICY System View

Presents a version of the table ISYSIQLSPOLICY in a readable format.

The ISYSIQLSPOLICY system table stores the logical server policies.

Column name	Column type	Description
ls_policy_Id	UNSIGNED BIGINT NOT NULL	The ID number of the logical server policy.
ls_policy_name	CHAR(128) NOT NULL UNIQUE	The logical server policy name.

Constraints on underlying system table:

- Primary key(ls_policy_id)
- object_id foreign key(ISYSOBJECT)

SYSIQLSPOLICYOPTION System View

Presents a version of the table ISYSIQLSPOLICYOPTION in a readable format.

The ISYSIQLSPOLICYOPTION table stores the logical server policy options.

Column name	Column type	Description
ls_policy_id	UNSIGNED BIGINT NOT NULL	The ID number of the login policy.
ls_policy_option_name	CHAR(128) NOT NULL	The logical server policy option name.
ls_policy_option_value	LONG VARCHAR NOT NULL	The logical server policy option value.

Constraints on underlying system table:

- Primary key(ls_policy_id, ls_policy_option_name)
- ls_policy_id foreign key(ISYSIQLSPOLICY)

SYSIQMPXSERVER System View

Presents a readable version of the table `ISYSIQMPXSERVER`. The ISYSIQMPXSERVER system table stores membership properties and version status data for the given multiplex node.

Column name	Column type	Description
server_id	UNSIGNED INT NOT NULL	The ID number of the server.
server_name	CHAR(128) NOT NULL	The server name. Must be case insensitive unique.
role	TINYINT NOT NULL	Coordinator, reader, or writer.
status	TINYINT NOT NULL	Excluded or included.
current_version	UNSIGNED BIGINT NULL	Current version ID of the server.
active_version	LONG BINARY NULL	The list of active versions on the server (encoded).
connection_info	LONG VARCHAR NULL	String containing host name and port pairs for public domain connections, delimited by semicolons.
db_path	LONG VARCHAR NOT NULL	Full path to the database file for the server.
private_connection_info	LONG VARCHAR NULL	String containing host name and port pairs for private network connections, delimited by semicolons.

Constraints on underlying system table:

- Primary key(server_id)

SYSIQOBJECTS ASE Compatibility View

sysiqobjects presents one row for each system table, user table, view, procedure, trigger, event, constraint, domain (sysdomain), domain (sysusertype), column, and index. This view is owned by user DBO.

See also

- *Tables in Each Adaptive Server Enterprise Database* on page 810
- *SYSCOLUMNS ASE Compatibility View* on page 703
- *SYSCOMMENTS ASE Compatibility View* on page 703
- *SYSINDEXES ASE Compatibility View* on page 726
- *SYSIQVINDEX ASE Compatibility View* on page 742
- *SYSOBJECTS ASE Compatibility View* on page 750
- *SYSTYPES ASE Compatibility View* on page 797
- *SYSUSERS ASE Compatibility View* on page 805

SYSIQPARTITIONCOLUMN System View

Presents group information from `ISYSIQPARTITIONCOLUMN` in a readable format.

```
ALTER VIEW "SYS"."SYSIQPARTITIONCOLUMN"
as select * from SYS.ISYSIQPARTITIONCOLUMN
```

Each row in the `SYSIQPARTITIONCOLUMN` view describes a column in a partition described in the `SYSIQPARTITION` view in a partitioned table described in the `SYSPARTITIONSCHEME` view. SYSIQPARTITIONCOLUMN only describes partitions of columns that are not stored on the dbspace of the partition.

Column name	Column type	Description
partitioned_object_id	unsigned bigint	Unique ID assigned to each partitioned object (table)
partition_id	unsigned int	Identifies a partition in a partitioned table.
column_id	unsigned int	The column ID of the column.
dbspace_id	smallint	The dbspace ID of the dbspace where this column of the partition is stored.

Constraints on underlying system table:

Primary key (partitioned_object_id, partition_id, column_id)

Foreign key (partitioned_object_id, partition_id) references SYS.ISYSPARTITION

Foreign key (dbspace_id) references SYS.ISYSDBSPACE

SYSIQRLVMERGEHISTORY System View

A log entry is added for each row-level versioning (RLV) enabled-table each time a merge between the RLV store and the IQ main store begins. Log entries are updated when the merge is complete.

Column Name	Column Type	Description
merge_id	unsigned bigint	Unique log entry identifier
table_id	unsigned int	Foreign key to the sys.systable system table
start_time	timestamp	Time the merge started
end_time	timestamp	Time the merge ended
status	char (9)	STARTED | COMPLETED | FAILED
return_code	tinyint	SQL code of the merge once completed
merge_type	char (9)	The cause of the merge trigger: AUTOMATIC | DML | DDL | SHUTDOWN | USER
merge_mode	char (12)	BLOCKING | NON-BLOCKING
merge_detail	varchar (255)	Additional information, if provided, such as error information
rows_inserted	unsigned bigint	Number of rows that were inserted as a result of the merge
rows_updated	unsigned bigint	Number of rows that were updated as a result of the merge
rows_deleted	unsigned bigint	Number of rows that were deleted as a result of the merge
rows_forwarded	unsigned bigint	Number of rows that were uncommitted at the time of the merge

SYSIQRVLOG System View

Presents group information from `ISYSIQRVLOG` in a readable format. Each row in the SYSIQRVLOG view corresponds to a log for a RLV-enabled table . The row with table_id 0 represents the server-wide commit log.

Column Name	Column Type	Description
stream_id	unsigned int	The log stream identifier.
table_id	unsigned int	Indicates the table the log stream belongs to. NULL indicates a commit log stream.
partition_low	unsigned int	Corresponds to the partition map in use when that log was last active.
partition_high	int	Corresponds to the partition map in use when that log was last active.
identity_location	unsigned bigint	Location of the log stream identity block.

SYSIQTAB System View

Presents group information from `ISYSIQTAB` in a readable format. Each row in the SYSIQTAB view describes an IQ table.

```
ALTER VIEW "SYS"."SYSIQTAB"
as select * from SYS.ISYSIQTAB
```

Note: This view replaces the deprecated system view SYSIQTABLE.

Column name	Column type	Description
table_id	unsigned int	Each table is assigned a unique number (the table number) that is the primary key.
block_map	hs_blockmapidentity	For internal use.
block_map_size	unsigned int	For internal use.
vdo	hs_vdoidentity	For internal use.
vdoid_size	unsigned int	For internal use.
info_location	hs_vdorecid	Not used. Always zero.
info_recid_size	unsigned int	Not used. Always zero.

Column name	Column type	Description
info_location_size	unsigned int	Not used. Always zero.
commit_txn_id	unsigned bigint	For internal use.
txn_id	unsigned bigint	For internal use.
update_time	timestamp	Last date and time the IQ table was modified.
is_rlv	char(1)	'T' – RLV storage is enabled on the table; 'F' – RLV storage is not enabled on the table.

Constraints on underlying system table:

Primary key (table_id)

See also

- *SYSIQTABLE System View (Deprecated)* on page 742

SYSIQTABCOL System View

Presents group information from `ISYSIQTABCOL` in a readable format. Each row in the SYSIQTABCOL view describes a column in an IQ table.

```
ALTER VIEW "SYS"."SYSIQTABCOL"
as select * from SYS.ISYSIQTABCOL
```

Note: This view replaces the deprecated system view SYSIQCOLUMN.

Column name	Column type	Description
link_table_id	unsigned int	For internal use.
link_column_id	unsigned int	For internal use.
max_length	unsigned int	Indicates the maximum length allowed by the column.
approx_unique_count	rowid	Approximate number of unique values (cardinality) of this column.
cardinality	rowid	The actual number of unique values (cardinality) of this column.
has_data	char(1)	Indicates that the column contains data (T/F).

Column name	Column type	Description
is_nbit	char(1)	Indicates whether the column is NBit (T) or Flat FP (F).

Constraints on underlying system table:

Primary key (table_id)

See also

- *SYSIQCOLUMN System View (Deprecated)* on page 729

SYSIQTABLE System View (Deprecated)

SYSIQTABLE has been replaced by the SYSIQTAB system view.

See also

- *SYSIQTAB System View* on page 740

SYSIQVINDEX ASE Compatibility View

sysiqvindex provides one row for each non-FP IQ index.

This view is owned by user DBO.

See also

- *Tables in Each Adaptive Server Enterprise Database* on page 810
- *SYSCOLUMNS ASE Compatibility View* on page 703
- *SYSCOMMENTS ASE Compatibility View* on page 703
- *SYSINDEXES ASE Compatibility View* on page 726
- *SYSIQOBJECTS ASE Compatibility View* on page 738
- *SYSOBJECTS ASE Compatibility View* on page 750
- *SYSTYPES ASE Compatibility View* on page 797
- *SYSUSERS ASE Compatibility View* on page 805

SYSIXCOL compatibility view (deprecated)

The SYSIXCOL view is provided for compatibility with older versions of the software that offered a SYSIXCOL system table. However, the SYSIXCOL system table has been replaced by the ISYSIDXCOL system table, and its corresponding SYSIDXCOL system view. You should switch to using the SYSIDXCOL system view.

Each row of the SYSIXCOL describes a column in an index. The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSIXCOL"
  as select a.table_id,
    a.index_id,
    a.sequence,
    a.column_id,
    a."order"
    from SYS.ISYSIDXCOL as a
      ,SYS.ISYSIDX as b
    where a.table_id = b.table_id
    and a.index_id = b.index_id
    and b.index_category = 3
```

SYSJAR system view

Each row in the SYSJAR system view defines a JAR file stored in the database. The underlying system table for this view is ISYSJAR.

Column name	Data type	Description
jar_id	INTEGER	A unique number identifying the JAR file.
object_id	UNSIGNED BIGINT	The internal ID for the JAR file, uniquely identifying it in the database.
creator	UNSIGNED INT	The user number of the creator of the JAR file. Can be set by the AS USER clause of the INSTALL JAVA statement.
jar_name	LONG VARCHAR	The name of the JAR file.
jar_file	LONG VARCHAR	This column is no longer used and contains NULL.
update_time	TIMESTAMP	The local time the JAR file was last updated.
update_time_utc	TIMESTAMP WITH TIME ZONE	The UTC time the JAR file was last updated.

Constraints on underlying system table

```
PRIMARY KEY (jar_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
UNIQUE INDEX (jar_name)
```

SYSJARCOMPONENT system view

Each row in the SYSJAR system view defines a JAR file component. The underlying system table for this view is ISYSJARCOMPONENT.

Column name	Data type	Description
component_id	INTEGER	The primary key containing the id of the component.
jar_id	INTEGER	A field containing the ID number of the JAR.
component_name	LONG VARCHAR	The name of the component.
component_type	CHAR(1)	This column is no longer used and contains NULL.
contents	LONG BINARY	The byte code of the JAR file.

Constraints on underlying system table

```
PRIMARY KEY (component_id)
```

```
FOREIGN KEY (jar_id) REFERENCES SYS.ISYSJAR (jar_id)
```

SYSJAVACLASS system view

Each row in the SYSJAVACLASS system view describes one Java class stored in the database. The underlying system table for this view is ISYSJAVACLASS.

Column name	Data type	Description
class_id	INTEGER	The unique number for the Java class. Also the primary key for the table.
object_id	UNSIGNED BIGINT	The internal ID for the Java class, uniquely identifying it in the database.
creator	UNSIGNED INT	The user number of the creator of the class. Can be set by the AS USER clause of the INSTALL JAVA statement.
jar_id	INTEGER	The id of the JAR file from which the class came.
class_name	LONG VARCHAR	The name of the Java class.
public	CHAR(1)	Indicates whether the class is public (Y) or private (N).

Column name	Data type	Description
component_id	INTEGER	The id of the component in the SYSJARCOMPONENT system view.
update_time	TIMESTAMP	The local last update time of the class.
update_time_utc	TIMESTAMP WITH TIME ZONE	The UTC last update time of the class.

Constraints on underlying system table

```
PRIMARY KEY (class_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
```

```
FOREIGN KEY (component_id) REFERENCES SYS.ISYSJARCOMPONENT
(component_id)
```

SYSLDAPSERVER System View

Presents information on the **ISYSLDAPSERVER** system table in a readable format.

The ISYSLDAPSERVER system table defines a set of attributes for the LDAP server.

Column name	Column type	Description
ldsrv_id	UNSIGNED BIGINT NOT NULL	A unique identifier for the LDAP server that is the primary key and is used by the login policy to refer to the LDAP server.
ldsrv_name	CHAR(128) NOT NULL	The name assigned to the LDAP server.
ldsrv_state	CHAR(9) NOT NULL	Read-only state of the LDAP server: 1 - RESET 2 - READY 3 - ACTIVE 4 - FAILED 5 - SUSPENDED **Note:** A numeric value is stored in system table; a corresponding text value appears in the system view.

Column name	Column type	Description
ldsrv_start_tls	TINYINT NOT NULL	Controls whether Transport Layer Security (TLS) is used to connect to the LDAP server. This provides encrypted communication for connections and searches with the LDAP server in conjunction with TRUSTED_CERITIFCATE_FILE. Valid range: 1 (ON) or 0 (OFF). Default value is 0 (OFF).
ldsrv_num_retries	TINYINT NOT NULL	Controls the number of authenticate attempts allowed with the LDAP server before returning a failure or initiating a failover (if specified). Valid range: 1-60 Default value is 3.
ldsrv_timeout	UNSIGNED INT NOT NULL	Controls the timeout value (in milliseconds) for connections or searches. Valid range: 1-3600000 (1 hour) Default value is 10000.
ldsrv_last_state_change	TIMESTAMP NOT NULL	Indicates the time the last state change occurred. The value is stored in Coordinated Universal Time (UTC), regardless of the local time zone of the LDAP server.
ldsrv_search_url	CHAR(1024) NULL	The LDAP URL to be used to find the Distinguished Name (DN) for a user based on their user ID.
ldsrv_auth_url	CHAR(1024) NULL	The LDAP search string to be used to find the DN for a user given their user ID.
ldsrv_access_dn	CHAR(1024) NULL	The DN used to access the LDAP server for searches to obtain the DN for a user ID.
ldsrv_access_dn_pwd	VARBINARY(1024) NULL	The password for the access account. The password is symmetrically encrypted when stored on disk.

SYSLOGINMAP system view

The SYSLOGINMAP system view contains one row for each user that can connect to the database using either an integrated login, or Kerberos login. For that reason, access to this view is restricted. The underlying system table for this view is ISYSLOGINMAP.

Column name	Data type	Description
login_mode	TINYINT	The type of login: 1 for integrated logins, 2 for Kerberos logins.
login_id	VARCHAR(1024)	Either the integrated login user profile name, or the Kerberos principal that maps to database_uid.
object_id	UNSIGNED BIGINT	A unique identifier, one for each mapping between user ID and database user ID.
database_uid	UNSIGNED INT	The database user ID to which the login ID is mapped.

Constraints on underlying system table

```
PRIMARY KEY (login_mode, login_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
FOREIGN KEY (database_uid) REFERENCES SYS.ISYSUSER (user_id)
```

SYSLOGINPOLICY system view

The underlying system table for this view is ISYSLOGINPOLICY.

Column name	Data type	Description
login_policy_id	UNSIGNED BIGINT	A unique identifier for the login policy.
login_policy_name	CHAR(128)	The name of the login policy.

Constraints on underlying system table

```
PRIMARY KEY (login_policy_id)
```

```
FOREIGN KEY (login_policy_id) REFERENCES SYS.ISYSOBJECT (object_id)
```

```
UNIQUE INDEX (login_policy_name)
```

SYSLOGINPOLICYOPTION system view

The underlying system table for this view is ISYSLOGINPOLICYOPTION.

Column name	Data type	Description
login_policy_id	UNSIGNED BIGINT	A unique identifier for the login policy.
login_option_name	CHAR(128)	The name of the login policy.
login_option_value	LONG VARCHAR	The value of the login policy at the time it was created.

Constraints on underlying system table

```
PRIMARY KEY (login_policy_id, login_option_name)
```

```
FOREIGN KEY (login_policy_id) REFERENCES SYS.ISYSLOGINPOLICY
(login_policy_id)
```

SYSLOGINS ASE Compatibility View

This view is owned by user DBO. SYSLOGINS contains one row for each valid Adaptive Server Enterprise user account.

SYSMVOPTION system view

Each row in the SYSMVOPTION system view describes the setting of one option value for a materialized view or text index at the time of its creation. The name of the option can be found in the SYSMVOPTIONNAME system view. The underlying system table for this view is ISYSMVOPTION.

Column name	Data type	Description
view_object_id	UNSIGNED BIGINT	The object ID of the materialized view.
option_id	UNSIGNED INT	A unique number identifying the option in the database. To see the option name, see the SYSMVOPTIONNAME system view.
option_value	LONG VARCHAR	The value of the option when the materialized view was created.

Constraints on underlying system table

```
PRIMARY KEY (view_object_id, option_id)
```

```
FOREIGN KEY (view_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
```

```
FOREIGN KEY (option_id) REFERENCES SYS.ISYSMVOPTIONNAME (option_id)
```

SYSMVOPTIONNAME system view

Each row in the SYSMVOPTION system view gives the name option value for a materialized view or text index at the time of its creation. The value for the option can be found in the SYSMVOPTION system view. The underlying system table for this view is ISYSMVOPTIONNAME.

Column name	Data type	Description
option_id	UNSIGNED INT	A number uniquely identifying the option in the database.
option_name	CHAR(128)	The name of the option.

Constraints on underlying system table

```
PRIMARY KEY (option_id)
```

```
UNIQUE INDEX (option_name)
```

SYSOBJECT system view

Each row in the SYSOBJECT system view describes a database object. The underlying system table for this view is ISYSOBJECT.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The internal ID for the object, uniquely identifying it in the database.

Column name	Data type	Description
status	TINYINT	The status of the object. Values include: • **1 (valid) –** The object is available for use by the database server. This status is synonymous with ENABLED. That is, if you ENABLE an object, the status changes to VALID. • **2 (invalid) –** An attempt to recompile the object after an internal operation has failed, for example, after a schema-altering modification to an object on which it depends. The database server continues to try to recompile the object whenever it is referenced in a statement. • **4 (disabled) –** The object has been explicitly disabled by the user, for example using an ALTER TABLE...DISABLE VIEW DEPENDENCIES statement.
object_type	TINYINT	Type of object.
creation_time	TIMESTAMP	The local date and time when the object was created.
object_type_str	CHAR (128)	Type of object.
creation_time_utc	TIMESTAMP WITH TIME ZONE	The UTC date and time when the object was created.

Constraints on underlying system table

```
PRIMARY KEY (object_id)
```

SYSOBJECTS ASE Compatibility View

sysobjects contains one row for each table, view, stored procedure, extended stored procedure, log, rule, default, trigger, check constraint, referential constraint, computed column, function-based index key, and temporary object, and other forms of compiled objects.

This view is owned by user DBO.

It also contains one row for each partition condition ID when object type is N.

See also

- *Tables in Each Adaptive Server Enterprise Database* on page 810
- *SYSCOLUMNS ASE Compatibility View* on page 703
- *SYSCOMMENTS ASE Compatibility View* on page 703
- *SYSINDEXES ASE Compatibility View* on page 726
- *SYSIQOBJECTS ASE Compatibility View* on page 738
- *SYSIQVINDEX ASE Compatibility View* on page 742
- *SYSTYPES ASE Compatibility View* on page 797
- *SYSUSERS ASE Compatibility View* on page 805

SYSOPTION system view

The SYSOPTION system view contains the options one row for each option setting stored in the database. Each user can have their own setting for a given option. In addition, settings for the PUBLIC role define the default settings to be used for users that do not have their own setting. The underlying system table for this view is ISYSOPTION.

Column name	Data type	Description
user_id	UNSIGNED INT	The user number to whom the option setting applies.
"option"	CHAR(128)	The name of the option.
"setting"	LONG VARCHAR	The current setting for the option.

Constraints on underlying system table

```
PRIMARY KEY (user_id, "option")
```

```
FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)
```

SYSOPTIONS consolidated view

Each row in the SYSOPTIONS view describes one option created using the SET command. Each user can have their own setting for each option. In addition, settings for the PUBLIC user define the default settings to be used for users that do not have their own setting.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSOPTIONS"( user_name,"option",setting )
  as select u.user_name,opt."option",opt.setting
    from SYS.ISYSOPTION as opt
      join SYS.ISYSUSER as u on opt.user_id = u.user_id
```

SYSOPTSTAT system view

The SYSOPTSTAT system view stores the cost model calibration information as computed by the ALTER DATABASE CALIBRATE statement. The contents of this view are for internal use only and are best accessed via the sa_get_dtt system procedure. The underlying system table for this view is ISYSOPTSTAT.

Column name	Data type	Description
stat_id	UNSIGNED INT	For system use only.
group_id	UNSIGNED INT	For system use only.
format_id	SMALLINT	For system use only.
data	LONG BINARY	For system use only.

Constraints on underlying system table

```
PRIMARY KEY (stat_id, group_id, format_id)
```

SYSPARTITION System View

Presents group information from `ISYSPARTITION` in a readable format.

Each row in the `SYSPARTITION` view describes a partitioned object (table or index) in the database.

```
ALTER VIEW "SYS"."SYSPARTITION"
as select * from SYS.ISYSPARTITION
```

Column name	Column type	Description
partitioned_object_id	unsigned bigint	Unique number assigned to each partitioned object (table)
partition_id	unsigned int	Identifies a partition in a partitioned table.
partition_object_id	unsigned bigint	Each table partition is an object itself and is assigned a unique number from the table object or index object.
partition_values	long varchar	Contains the upper bound for this range partition.
position	unsigned int	Ordinal number of partition.
partition_name	char(128)	Name of partition

Constraints on underlying system table:

Primary key (partitioned_object_id, partition_id)

Unique (partition_object_id, position)

Foreign key (partition_object_id) references SYS.ISYSOBJECT

Foreign key (partitioned_object_id) references SYS.ISYSOBJECT

SYSPARTITIONKEY System View

Presents group information from `ISYSPARTITIONKEY` in a readable format.

Each row in the `SYSPARTITIONKEY` view describes a partitioned object (table or index) in the database.

```
ALTER VIEW "SYS"."SYSPARTITIONKEY"
as select * from SYS.ISYSPARTITIONKEY
```

Column name	Column type	Description
partitioned_object_id	unsigned bigint	Each partitioned object (table) is assigned a unique object number.
column_id	unsigned int	The column ID identifies the table column as part of the partitioning key.
position	smallint	Position of this column in the partitioning key. Position is 0 based. A position of 0 indicates the 1st column in the partitioning key.

Constraints on underlying system table:

Primary key (partitioned_object_id, column_id)

Foreign key (partitioned_object_id) references SYS.ISYSOBJECT

SYSPARTITIONSCHEME System View

Presents group information from `ISYSPARTITIONSCHEME` in a readable format.

Each row in the `SYSPARTITIONSCHEME` view describes a partitioned object (table or index) in the database.

```
ALTER VIEW "SYS"."SYSPARTITIONSCHEME"
as select * from SYS.ISYSPARTITIONSCHEME
```

Column Name	Column Type	Description
partitioned_object_id	unsigned bigint	Each partitioned object (table) is assigned a unique number

Column Name	Column Type	Description
partition_method	tinyint	The partitioning method for this table. Valid values: 1– for range SAP Sybase IQ supports only range partitioning.
subpartition_method	tinyint	Reserved for future use.

Constraints on underlying system table:

Primary key (partitioned_object_id)

Foreign key (partitioned_object_id) references SYS.ISYSOBJECT

SYSPHYSIDX system view

Each row in the SYSPHYSIDX system view defines a physical index in the database. The underlying system table for this view is ISYSPHYSIDX.

Column name	Data type	Description
table_id	UNSIGNED INT	The object ID of the table to which the index corresponds.
phys_index_id	UNSIGNED INT	The unique number of the physical index within its table.
root	INTEGER	Identifies the location of the root page of the physical index in the database file.
key_value_count	UNSIGNED INT	The number of distinct key values in the index.
leaf_page_count	UNSIGNED INT	The number of leaf index pages.
depth	UNSIGNED SMALLINT	The depth (number of levels) of the physical index.
max_key_distance	UNSIGNED INT	For system use only.
seq_transitions	UNSIGNED INT	For system use only.
rand_transitions	UNSIGNED INT	For system use only.
rand_distance	UNSIGNED INT	For system use only.
allocation_bitmap	LONG VARBIT	For system use only.
long_value_bitmap	LONG VARBIT	For system use only.

Constraints on underlying system table

```
PRIMARY KEY (table_id, phys_index_id)
```

SYSPROCAUTH consolidated view

Each row in the SYSPROCAUTH view describes a set of privileges granted on a procedure. As an alternative, you can also use the SYSPROCPERM system view.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSPROCAUTH"( grantee,
  creator,procname )
  as select u1.user_name,u2.user_name,p.proc_name
    from SYS.ISYSPROCEDURE as p
      join SYS.ISYSPROCPERM as pp on(p.proc_id = pp.proc_id)
      join SYS.ISYSUSER as u1 on u1.user_id = pp.grantee
      join SYS.ISYSUSER as u2 on u2.user_id = p.creator
```

SYSPROCEDURE system view

Each row in the SYSPROCEDURE system view describes one procedure in the database. The underlying system table for this view is ISYSPROCEDURE.

Column name	Data type	Description
proc_id	UNSIGNED INT	Each procedure is assigned a unique number (the procedure number).
creator	UNSIGNED INT	The owner of the procedure.
object_id	UNSIGNED BIGINT	The internal ID for the procedure, uniquely identifying it in the database.
proc_name	CHAR(128)	The name of the procedure. One creator cannot have two procedures with the same name.
proc_defn	LONG VARCHAR	The definition of the procedure.
remarks	LONG VARCHAR	Remarks about the procedure. This value is stored in the ISYSREMARK system table.
replicate	CHAR(1)	This property is for internal use only.
srvid	UNSIGNED INT	If the procedure is a proxy for a procedure on a remote database server, indicates the remote server.

Column name	Data type	Description
source	LONG VARCHAR	The preserved source for the procedure. This value is stored in the ISYSSOURCE system table.
avg_num_rows	FLOAT	Information collected for use in query optimization when the procedure appears in the FROM clause.
avg_cost	FLOAT	Information collected for use in query optimization when the procedure appears in the FROM clause.
stats	LONG BINARY	Information collected for use in query optimization when the procedure appears in the FROM clause.

Constraints on underlying system table

```
PRIMARY KEY (proc_id)
```

```
FOREIGN KEY (srvid) REFERENCES SYS.ISYSSERVER (srvid)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
```

```
UNIQUE INDEX (proc_name, creator)
```

SYSPROCPARM system view

Each row in the SYSPROCPARM system view describes one parameter to a procedure in the database. The underlying system table for this view is ISYSPROCPARM.

Column name	Data type	Description
proc_id	UNSIGNED INT	Uniquely identifies the procedure to which the parameter belongs.

Column name	Data type	Description
parm_id	SMALLINT	Each procedure starts numbering parameters at 1. The order of parameter numbers corresponds to the order in which they were defined. For functions, the first parameter has the name of the function and represents the return value for the function.
parm_type	SMALLINT	The type of parameter is one of the following: • **0 –** Normal parameter (variable) • **1 –** Result variable - used with a procedure that returns result sets • **2 –** SQLSTATE error value • **3 –** SQLCODE error value • **4 –** Return value from function
parm_mode_in	CHAR(1)	Indicates whether the parameter supplies a value to the procedure (IN or INOUT parameters).
parm_mode_out	CHAR(1)	Indicates whether the parameter returns a value from the procedure (OUT or INOUT parameters) or columns in the RESULT clause.
domain_id	SMALLINT	Identifies the data type for the parameter, by the data type number listed in the SYSDOMAIN system view.
width	BIGINT	Contains the length of a string parameter, the precision of a numeric parameter, or the number of bytes of storage for any other data type.

Column name	Data type	Description
scale	SMALLINT	For numeric data types, the number of digits after the decimal point. For all other data types, the value of this column is 1.
user_type	SMALLINT	The user type of the parameter, if applicable.
parm_name	CHAR(128)	The name of the procedure parameter.
"default"	LONG VARCHAR	Default value of the parameter. Provided for informational purposes only.
remarks	LONG VARCHAR	Always returns NULL. Provided to allow the use of previous versions of ODBC drivers with newer personal database servers.
base_type_str	VARCHAR(32767)	The annotated type string representing the physical type of the parameter.

Constraints on underlying system table

```
PRIMARY KEY (proc_id, parm_id)
```

```
FOREIGN KEY (proc_id) REFERENCES SYS.ISYSPROCEDURE (proc_id)
```

```
FOREIGN KEY (domain_id) REFERENCES SYS.ISYSDOMAIN (domain_id)
```

```
FOREIGN KEY (user_type) REFERENCES SYS.ISYSUSERTYPE (type_id)
```

SYSPROCPARMS consolidated view

Each row in the SYSPROCPARMS view describes a parameter to a procedure in the database.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSPROCPARMS"( creator,
  procname,parmname,parm_id,parmtype,parmmode,parmdomain,
  length,scale,"default",user_type )
  as select
up.user_name,p.proc_name,pp.parm_name,pp.parm_id,pp.parm_type,
    if pp.parm_mode_in = 'Y' and pp.parm_mode_out = 'N' then 'IN'
    else if pp.parm_mode_in = 'N' and pp.parm_mode_out = 'Y' then
'OUT'
      else 'INOUT'
      endif

endif,dom.domain_name,pp.width,pp.scale,pp."default",ut.type_name
    from SYS.SYSPROCPARM as pp
      join SYS.ISYSPROCEDURE as p on p.proc_id = pp.proc_id
      join SYS.ISYSUSER as up on up.user_id = p.creator
      join SYS.ISYSDOMAIN as dom on dom.domain_id = pp.domain_id
      left outer join SYS.ISYSUSERTYPE as ut on ut.type_id =
pp.user_type
```

SYSPROCPERM system view

Each row of the SYSPROCPERM system view describes a user who has been granted EXECUTE privilege on a procedure. The underlying system table for this view is ISYSPROCPERM.

Column name	Data type	Description
proc_id	UNSIGNED INT	The procedure number uniquely identifies the procedure for which EXECUTE privilege has been granted.
grantee	UNSIGNED INT	The user number of the privilege grantee.

Constraints on underlying system table

```
PRIMARY KEY (proc_id, grantee)
```

```
FOREIGN KEY (grantee) REFERENCES SYS.ISYSUSER (user_id)
```

```
FOREIGN KEY (proc_id) REFERENCES SYS.ISYSPROCEDURE (proc_id)
```

SYSPROCS consolidated view

The SYSPROCS view shows the procedure or function name, the name of its creator and any comments recorded for the procedure or function.

The tables and columns that make up this view are provided in the ALTER VIEW statement below.

```
ALTER VIEW "SYS"."SYSPROCS"( creator,
  procname,remarks )
  as select u.user_name,p.proc_name,r.remarks
    from SYS.ISYSPROCEDURE as p
      join SYS.ISYSUSER as u on u.user_id = p.creator
      left outer join SYS.ISYSREMARK as r on(p.object_id =
r.object_id)
```

SYSPROXYTAB system view

Each row of the SYSPROXYTAB system view describes the remote parameters of one proxy table. The underlying system table for this view is ISYSPROXYTAB.

Column name	Data type	Description
table_object_id	UNSIGNED BIGINT	The object ID of the proxy table.
existing_obj	CHAR(1)	Indicates whether the proxy table previously existed on the remote server.
srvid	UNSIGNED INT	The unique ID for the remote server associated with the proxy table.
remote_location	LONG VARCHAR	The location of the proxy table on the remote server.

Constraints on underlying system table

```
PRIMARY KEY (table_object_id)
```

```
FOREIGN KEY (table_object_id) REFERENCES ISYSOBJECT (object_id)
MATCH UNIQUE FULL
```

```
FOREIGN KEY (srvid) REFERENCES SYS.ISYSSERVER (srvid)
```

SYSPUBLICATION system view

Each row in the SYSPUBLICATION system view describes a publication. The underlying system table for this view is ISYSPUBLICATION.

Column name	Data type	Description
publication_id	UNSIGNED INT	A number uniquely identifying the publication.
object_id	UNSIGNED BIGINT	The internal ID for the publication, uniquely identifying it in the database.
creator	UNSIGNED INT	The owner of the publication.
publication_name	CHAR(128)	The name of the publication.
remarks	LONG VARCHAR	Remarks about the publication. This value is stored in the ISYSREMARK system table.
type	CHAR(1)	This column is deprecated.

Column name	Data type	Description
sync_type	UNSIGNED INT	The type of synchronization for the publication. Values include: • **0 (logscan) –** This is a regular publication that uses the transaction log to upload all relevant data that has changed since the last upload. • **1 (scripted upload) –** For this publication, the transaction log is ignored and the upload is defined by the user using stored procedures. Information about the stored procedures is stored in the ISYSSYNCSCRIPT system table. • **2 (download only) –** This is a download-only publication; no data is uploaded.

Constraints on underlying system table

```
PRIMARY KEY (publication_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
```

```
UNIQUE INDEX (publication_name, creator)
```

SYSPUBLICATIONS consolidated view

Each row in the SYSPUBLICATIONS view describes a publication.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSPUBLICATIONS"
  as select u.user_name as creator,
    p.publication_name,
    r.remarks,
    p.type,
    case p.sync_type
    when 0 then 'logscan'
    when 1 then 'scripted upload'
    when 2 then 'download only'
    else 'invalid'
    end as sync_type
    from SYS.ISYSPUBLICATION as p
      join SYS.ISYSUSER as u on u.user_id = p.creator
      left outer join SYS.ISYSREMARK as r on(p.object_id =
r.object_id)
```

SYSREMARK system view

Each row in the SYSREMARK system view describes a remark (or comment) for an object. The underlying system table for this view is ISYSREMARK.

Column	Data type	Description
object_id	UNSIGNED BIGINT	The internal ID for the object that has an associated remark.
remarks	LONG VARCHAR	The remark or comment associated with the object.

Constraints on underlying system table

```
PRIMARY KEY (object_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

SYSREMOTEOPTION system view

Each row in the SYSREMOTEOPTION system view describes the value of a message link parameter. The underlying system table for this view is ISYSREMOTEOPTION.

Some columns in this view contain potentially sensitive data. The SYSREMOTEOPTION2 view provides public access to the data in this view except for the potentially sensitive columns.

Column	Data type	Description
option_id	UNSIGNED INT	An identification number for the message link parameter.
user_id	UNSIGNED INT	The user ID for which the parameter is set.
"setting"	VARCHAR(255)	The value of the message link parameter.

Constraints on underlying system table

```
PRIMARY KEY (option_id, user_id)
```

```
FOREIGN KEY (option_id) REFERENCES SYS.ISYSREMOTEOPTIONTYPE
(option_id)
```

```
FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)
```

SYSREMOTEOPTION2 consolidated view

Joins together, and presents in a more readable format, the columns from SYSREMOTEOPTION and SYSREMOTEOPTIONTYPE system views.

Values in the setting column are hidden from users that do not have the SELECT ANY TABLE system privilege.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOTEOPTION2"
  as select ISYSREMOTEOPTION.option_id,
    ISYSREMOTEOPTION.user_id,
    SYS.HIDE_FROM_NON_DBA(ISYSREMOTEOPTION.setting) as setting
    from SYS.ISYSREMOTEOPTION
```

SYSREMOTEOPTIONS consolidated view

Each row of the SYSREMOTEOPTIONS view describes the values of a message link parameter.

Values in the setting column are hidden from users that do not have the SELECT ANY TABLE system privilege. The SYSREMOTEOPTION2 view provides public access to the insensitive data.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOTEOPTIONS"
  as select srt.type_name,
    sup.user_name,
    srot."option",
    SYS.HIDE_FROM_NON_DBA(sro.setting) as setting
    from SYS.ISYSREMOTETYPE as srt
      ,SYS.ISYSREMOTEOPTIONTYPE as srot
      ,SYS.ISYSREMOTEOPTION as sro
      ,SYS.ISYSUSER as sup
    where srt.type_id = srot.type_id
    and srot.option_id = sro.option_id
    and sro.user_id = sup.user_id
```

SYSREMOTEOPTIONTYPE system view

Each row in the SYSREMOTEOPTIONTYPE system view describes one of the message link parameters. The underlying system table for this view is ISYSREMOTEOPTIONTYPE.

Column	Data type	Description
option_id	UNSIGNED INT	An identification number for the message link parameter.

Column	Data type	Description
type_id	SMALLINT	An identification number for the message type that uses the parameter.
option	VARCHAR(128)	The name of the message link parameter.

Constraints on underlying system table

```
PRIMARY KEY (option_id)
```

```
FOREIGN KEY (type_id) REFERENCES SYS.ISYSREMOTETYPE (type_id)
```

SYSREMOTETYPE system view

The underlying system table for this view is ISYSREMOTETYPE.

Column name	Data type	Description
type_id	SMALLINT	Identifies which of the message systems supported is to be used to send messages to the user.
object_id	UNSIGNED BIGINT	The internal ID for the remote type, uniquely identifying it in the database.
type_name	CHAR(128)	The name of the message system.
publisher_address	LONG VARCHAR	The address of the remote database publisher.
remarks	LONG VARCHAR	Remarks about the remote type. This value is stored in the ISYSREMARK system table.

Constraints on underlying system table

```
PRIMARY KEY (type_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
UNIQUE INDEX (type_name)
```

SYSREMOTETYPES consolidated view

Each row of the SYSREMOTETYPES view describes one of the message types, including the publisher address.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOTETYPES"
  as select rt.type_id,rt.type_name,rt.publisher_address,rm.remarks
    from SYS.ISYSREMOTETYPE as rt
      left outer join SYS.ISYSREMARK as rm on(rt.object_id =
rm.object_id)
```

SYSREMOTEUSER system view

Each row in the SYSREMOTEUSER system view describes a user ID with the REMOTE system privilege (a subscriber), together with the status of messages that were sent to and from that user. The underlying system table for this view is ISYSREMOTEUSER.

Column name	Data type	Description
user_id	UNSIGNED INT	The user number of the user with REMOTE privilege.
consolidate	CHAR(1)	Indicates whether the user was granted CONSOLIDATE privilege (Y) or REMOTE privileges (N).
type_id	SMALLINT	Identifies which of the message systems is used to send messages to the user.
address	LONG VARCHAR	The address to which messages are to be sent. The address must be appropriate for the address_type.
frequency	CHAR(1)	How frequently messages are sent.
send_time	TIME	The next time messages are to be sent to this user.
log_send	UNSIGNED BIGINT	Messages are sent only to subscribers for whom log_send is greater than log_sent.
time_sent	TIMESTAMP	The local time the most recent message was sent to this subscriber.

Column name	Data type	Description
log_sent	UNSIGNED BIGINT	The log offset for the most recently sent operation.
confirm_sent	UNSIGNED BIGINT	The log offset for the most recently confirmed operation from this subscriber.
send_count	INTEGER	How many messages have been sent.
resend_count	INTEGER	Counter to ensure that messages are applied only once at the subscriber database.
time_received	TIMESTAMP	The local time when the most recent message was received from this subscriber.
log_received	UNSIGNED BIGINT	The log offset in the database of the subscriber for the operation that was most recently received at the current database.
confirm_received	UNSIGNED BIGINT	The log offset in the database of the subscriber for the most recent operation for which a confirmation message has been sent.
receive_count	INTEGER	How many messages have been received.
rereceive_count	INTEGER	Counter to ensure that messages are applied only once at the current database.
time_sent_utc	TIMESTAMP WITH TIME ZONE	The UTC time the most recent message was sent to this subscriber.
time_received_utc	TIMESTAMP WITH TIME ZONE	The UTC time when the most recent message was received from this subscriber.

Constraints on underlying system table

```
PRIMARY KEY (user_id)
```

```
FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)
```

```
FOREIGN KEY (type_id) REFERENCES SYS.ISYSREMOTETYPE (type_id)
```

```
UNIQUE INDEX (type_id, address)
```

SYSREMOTEUSERS consolidated view

Each row of the SYSREMOTEUSERS view describes a user ID with the REMOTE system privilege (a subscriber), together with the status of messages that were sent to and from that user.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOTEUSERS" AS SELECT u.user_name,
r.consolidate, t.type_name, r.address, r.frequency, r.send_time,
(if r.frequency = 'A' then NULL
else if r.frequency = 'P' then
if r.time_sent IS NULL then CURRENT TIMESTAMP
else (select min( minutes( dateadd(mi,
PROPERTY('TimeZoneAdjustment'), a.time_sent),
60*hour(a.send_time) + minute( seconds( a.send_time, 59 ) ) ) )
FROM SYS.ISYSREMOTEUSER a WHERE a.frequency = 'P' AND a.send_time =
r.send_time ) endif
else if CURRENT DATE + r.send_time > coalesce( dateadd(mi,
PROPERTY('TimeZoneAdjustment'), r.time_sent), CURRENT TIMESTAMP)
then CURRENT DATE + r.send_time
else CURRENT DATE + r.send_time + 1
endif   endif  endif) as next_send, r.log_send ,
dateadd(mi, PROPERTY('TimeZoneAdjustment'), r.time_sent)
as time_sent , r.log_sent, r.confirm_sent, r.send_count,
r.resend_count,
dateadd(mi, PROPERTY('TimeZoneAdjustment'), r.time_received) as
time_received ,
r.log_received, r.confirm_received, r.receive_count,
r.rereceive_count ,
TODATETIMEOFFSET( r.time_sent, 0 ) as time_sent_utc ,
TODATETIMEOFFSET( r.time_received, 0 ) as time_received_utc
FROM SYS.ISYSREMOTEUSER r JOIN SYS.ISYSUSER u ON ( u.user_id =
r.user_id ) JOIN SYS.ISYSREMOTETYPE t ON ( t.type_id = r.type_id )
```

SYSROLEGRANTEXT System View

The SYSROLEGRANTEXT system view contains syntax extensions pertaining to the SET USER and CHANGE PASSWORD system privilege and is related to the SYSROLEGRANT system view.

Column name	Data type	Description
grant_id	UNSIGNED INT	ID used to identify each GRANT statement.
user_id	UNSIGNED INT	The user_ids specified in `user-list` or `role-list` in a particular extended grant.

When you grant or revoke the SET USER or CHANGE PASSWORD privilege, either with the user-list option or with ANY WITH ROLES role-list option, this view is updated with the values from the extended syntax.

Constraints on underlying system table
PRIMARY KEY (grant_id, user_id)

SYSROLEGRANT System View

The SYSROLEGRANT system view contains one row for each grant of a system or user defined role. The underlying system table for this view is ISYSROLEGRANT.

Column name	Data type	Description
grant_id	UNSIGNED INT	ID used to identify each GRANT statement.
role_id	UNSIGNED INT	ID of the role being granted, as per ISYSUSER.
grantee	UNSIGNED INT	ID of the user being granted the role, as per ISYSUSER.
grant_type	TINYINT	Describes type of grant using 3 digits. The first digit is whether privilege has been granted. The second digit is whether administration rights have been given. The third digit is whether system privileges are inheritable. • **001 –** Privilege granted, with no inheritance, and no administration rights. Applicable only for legacy non-inheritable authorities except SYS_AUTH_DBA_ROLE and SYS_AUTH_RE-MOVE_DBA_ROLE • **101 –** Privilege granted, with inheritance, but no administration rights. • **110 –** Only administration rights have been granted. • **111 –** Privilege granted, with inheritance, and with administration rights • **001 –** Privilege granted, with administration rights, but no inheritance. Applicable only for legacy authorities SYS_AUTH_DBA_ROLE and SYS_AUTH_REMOVE_DBA_ROLE.

Column name	Data type	Description
grant_scope	TINYINT	Used by SET USER and CHANGE PASSWORD to set the scope of the grant. Values can be one or more of the following: • **001 –** User list. • **010 –** ANY WITH ROLES • **110 –** ANY
grantor	CHAR (128)	The unique identifier of the grantor of the role.

Constraints on underlying system table
PRIMARY KEY (grant_ID)

UNIQUE Index (role_id, grantee, grant_scope)

SYSROLEGRANTS System View

The SYSROLEGRANTS system view is the same as the SYSROLEGRANT system view but includes two additional columns: the name of the role (not just the role ID) and the name of the grantee (not just user ID).

Column name	Data type	Description
grant_id	UNSIGNED INT	A unique identifier for each grant statement issued.
role_id	UNSIGNED INT	The unique identifier for the role granted to a user (as defined in the ISYSUSER table).
role_name	CHAR(128)	The name of the role corresponding to the role_id value.
grantee	UNSIGNED INT	The unique identifier for the user granted the role.
grantee_name	CHAR(128)	The name of the grantee corresponding to the grantee value.

Column name	Data type	Description
grant_type	TINYINT	Identifies how the role and its underlying privileges were granted. Values include: • 1 - Underlying privileges are granted with no administrative rights and no privilege inheritance. **Note:** This value is applicable to all legacy, non-inheritable roles EXCEPT SYS_AUTH_DBA_ROLE and SYS_AUTH_REMOVE_DBA_ROLE. • 3 - Underlying privileges are granted with administrative rights, but with no privilege inheritance. **Note:** This value is applicable only to the legacy, non-inheritable roles SYS_AUTH_DBA_ROLE and SYS_AUTH_REMOVE_DBA_ROLE. • 5 - Underlying privileges are granted with no administrative rights, but with privilege inheritance. • 6 - Only administrative rights to the underlying privileges are granted. • 7 - Underlying privilege are granted with administrative rights and privilege inheritance. •
grant_scope	TINYINT	Defines the range to which the grant applies. Values include: • 1 - User list • 2 - Any users granted membership in the specified roles • 3 - All users **Note:** This value is applicable to the SET USER and CHANGE PASSWORD system privileges only and can store be any valid combination of these values.
grantor	CHAR (128)	The unique identifier of the grantor of the role.

SYSSCHEDULE system view

Each row in the SYSSCHEDULE system view describes a time at which an event is to fire, as specified by the SCHEDULE clause of CREATE EVENT. The underlying system table for this view is ISYSSCHEDULE.

Column name	Data type	Description
event_id	UNSIGNED INT	The unique number assigned to each event.
sched_name	VARCHAR(128)	The name associated with the schedule for the event.
recurring	TINYINT	Indicates if the schedule is re- peating.
start_time	TIME	The schedule start time.
stop_time	TIME	The schedule stop time if BE- TWEEN was used.
start_date	DATE	The first date on which the event is scheduled to execute.

days_of_week	TINYINT	A bit mask indicating the days of the week on which the event is scheduled:
		x01 = Sunday
		x02 = Monday
		x04 = Tuesday
		x08 = Wednesday
		x10 = Thursday
		x20 = Friday
		x40 = Saturday

days_of_month	UNSIGNED INT	A bit mask indicating the days of the month on which the event is scheduled. Some examples include:
		x01 = first day
		x02 = second day
		x40000000 = 31st day
		x80000000 = last day of month

Column name	Data type	Description

interval_units	CHAR(10)	The interval unit specified by EVERY:
		HH = hours
		NN = minutes
		SS = seconds

interval_amt	INTEGER	The period specified by EVERY.

Constraints on underlying system table

```
PRIMARY KEY (event_id, sched_name)
```

```
FOREIGN KEY (event_id) REFERENCES SYS.ISYSEVENT (event_id)
```

SYSSERVER system view

Each row in the SYSSERVER system view describes a remote server. The underlying system table for this view is ISYSSERVER.

Note: Previous versions of the catalog contained a SYSSERVERS system table. That table has been renamed to be ISYSSERVER (without an 'S'), and is the underlying table for this view.

Column name	Data type	Description
srvid	UNSIGNED INT	An identifier for the remote server.
srvname	VARCHAR(128)	The name of the remote server.
srvclass	LONG VARCHAR	The server class, as specified in the CREATE SERVER statement.
srvinfo	LONG VARCHAR	Server information.
srvreadonly	CHAR(1)	Whether the server is read-only.

Constraints on underlying system table

```
PRIMARY KEY (srvid)
```

SYSSOURCE system view

Each row in the SYSSOURCE system view contains the source code, if applicable, for an object listed in the SYSOBJECT system view. The underlying system table for this view is ISYSSOURCE.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The internal ID for the object whose source code is being defined.
source	LONG VARCHAR	This column contains the original source code for the object if the preserve_source_format database option is On when the object was created.

Constraints on underlying system table

```
PRIMARY KEY (object_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

SYSSQLSERVERTYPE system view

The SYSSQLSERVERTYPE system view contains information relating to compatibility with Adaptive Server Enterprise. The underlying system table for this view is ISYSSQLSERVERTYPE.

Column name	Data type	Description
ss_user_type	SMALLINT	The Adaptive Server Enterprise user type.
ss_domain_id	SMALLINT	The Adaptive Server Enterprise domain ID.
ss_type_name	VARCHAR (30)	The Adaptive Server Enterprise type name.
primary_sa_domain_id	SMALLINT	The corresponding SAP Sybase IQ primary domain ID.
primary_sa_user_type	SMALLINT	The corresponding SAP Sybase IQ primary user type.

Constraints on underlying system table

```
PRIMARY KEY (ss_user_type)
```

SYSSUBPARTITIONKEY System View

This view is reserved for future use. SAP Sybase IQ 16.0 does not support subpartitioning.

SYSSUBSCRIPTION system view

Each row in the SYSSUBSCRIPTION system view describes a subscription from one user ID (which must have the REMOTE system privilege) to one publication. The underlying system table for this view is ISYSSUBSCRIPTION.

Column name	Data type	Description
publication_id	UNSIGNED INT	The identifier for the publication to which the user ID is subscribed.
user_id	UNSIGNED INT	The ID of the user who is subscribed to the publication.
subscribe_by	CHAR(128)	The value of the SUBSCRIBE BY expression, if any, for the subscription.
created	UNSIGNED BIGINT	The offset in the transaction log at which the subscription was created.
started	UNSIGNED BIGINT	The offset in the transaction log at which the subscription was started.

Constraints on underlying system table

```
PRIMARY KEY (publication_id, user_id, subscribe_by)
```

```
FOREIGN KEY (publication_id) REFERENCES SYS.ISYSPUBLICATION
(publication_id)
```

```
FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)
```

SYSSUBSCRIPTIONS consolidated view

Each row describes a subscription from one user ID (which must have the REMOTE system privilege) to one publication.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSSUBSCRIPTIONS"
  as select p.publication_name,u.user_name,s.subscribe_by,s.created,
    s.started
    from SYS.ISYSSUBSCRIPTION as s
      join SYS.ISYSPUBLICATION as p on(p.publication_id =
s.publication_id)
      join SYS.ISYSUSER as u on u.user_id = s.user_id
```

SYSSYNC system view

The SYSSYNC system view contains information relating to synchronization. Some columns in this view contain potentially sensitive data. For that reason, access to this view is restricted. The SYSSYNC2 view provides public access to the data in this view except for the potentially sensitive columns. The underlying system table for this view is ISYSSYNC.

Column name	Data type	Description
sync_id	UNSIGNED INT	A number that uniquely identifies the row.
type	CHAR(1)	This value is always D.
publication_id	UNSIGNED INT	A publication_id found in the SYSPUBLICATION system view.
progress	UNSIGNED BIGINT	The log offset of the last successful upload.
site_name	CHAR(128)	A user name.
"option"	LONG VARCHAR	Synchronization options.
server_connect	LONG VARCHAR	The address or URL of the server.
server_conn_type	LONG VARCHAR	The communication protocol, such as TCP/IP, to use when synchronizing.
last_download_time	TIMESTAMP	Indicates the last time a download stream was received from the server.
last_upload_time	TIMESTAMP	Indicates the last time (measured at the server) that information was successfully uploaded. The default is jan-1-1900.
created	UNSIGNED BIGINT	The log offset at which the subscription was created.
log_sent	UNSIGNED BIGINT	The log progress up to which information has been uploaded. It is not necessary that an acknowledgement of the upload be received for the entry in this column to be updated.

Column name	Data type	Description
generation_number	INTEGER	For file-base downloads, the last generation number received for this subscription. The default is 0.
extended_state	VARCHAR(1024)	For internal use only.
script_version	CHAR(128)	Indicates the script version used by the CREATE and ALTER SYNCHRONIZATION SUBSCRIPTION statements and the START SYNCHRONIZATION SCHEMA CHANGE statement.
subscription_name	CHAR (128)	The name of the subscription.
server_protocol	UNSIGNED BIGINT	For internal use only.

Constraints on underlying system table

```
PRIMARY KEY (sync_id)
```

```
FOREIGN KEY (publication_id) REFERENCES SYS.ISYSPUBLICATION
(publication_id)
```

```
UNIQUE INDEX (publication_id, site_name)
```

```
UNIQUE INDEX (subscription_name)
```

SYSSYNC2 consolidated view

The SYSSYNC2 view provides public access to the data found in the SYSSYNC system view—information relating to synchronization—without exposing potentially sensitive data.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

Values in the server_connect and option columns are hidden from users that do not have the SELECT ANY TABLE system privilege.

```
ALTER VIEW "SYS"."SYSSYNC2"
  as select ISYSSYNC.sync_id,
    ISYSSYNC.type,
    ISYSSYNC.publication_id,
    ISYSSYNC.progress,
    ISYSSYNC.site_name,
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC."option") as "option",
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC.server_connect) as
server_connect,
    ISYSSYNC.server_conn_type,
    ISYSSYNC.last_download_time,
    ISYSSYNC.last_upload_time,
    ISYSSYNC.created,
    ISYSSYNC.log_sent,
    ISYSSYNC.generation_number,
    ISYSSYNC.extended_state,
    ISYSSYNC.script_version,
    ISYSSYNC.subscription_name
    from SYS.ISYSSYNC
```

SYSSYNCPUBLICATIONDEFAULTS consolidated view

The SYSSYNCPUBLICATIONDEFAULTS view provides the default synchronization settings associated with publications involved in synchronization.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

Values in the server_connect and option columns are hidden from users that do not have the SELECT ANY TABLE system privilege.

```
ALTER VIEW "SYS"."SYSSYNCPUBLICATIONDEFAULTS"
  as select s.sync_id,
    p.publication_name,
    SYS.HIDE_FROM_NON_DBA(s."option") as "option",
    SYS.HIDE_FROM_NON_DBA(s.server_connect) as server_connect,
    s.server_conn_type
    from SYS.ISYSSYNC as s join SYS.ISYSPUBLICATION as p
on(p.publication_id = s.publication_id) where
    s.site_name is null
```

SYSSYNCS consolidated view

The SYSSYNCS view contains information relating to synchronization.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

Values in the server_connect and option columns are hidden from users that do not have the SELECT ANY TABLE system privilege.

```
ALTER VIEW "SYS"."SYSSYNCS"
  as select p.publication_name,s.progress,s.site_name,
    SYS.HIDE_FROM_NON_DBA(s."option") as "option",
    SYS.HIDE_FROM_NON_DBA(s.server_connect) as server_connect,
    s.server_conn_type,s.last_download_time,
    s.last_upload_time,s.created,s.log_sent,s.generation_number,
    s.extended_state
    from SYS.ISYSSYNC as s
      left outer join SYS.ISYSPUBLICATION as p
      on p.publication_id = s.publication_id
```

SYSSYNCSCRIPT system view

Each row in the SYSSYNCSCRIPT system view identifies a stored procedure for scripted upload. This view is almost identical to the SYSSYNCSCRIPTS view, except that the values in this view are in their raw format.

The underlying system table for this view is ISYSSYNCSCRIPT.

Column name	Data type	Description
pub_object_id	UNSIGNED BIGINT	The object ID of the publication to which the script belongs.
table_object_id	UNSIGNED BIGINT	The object ID of the table to which the script applies.
type	UNSIGNED INT	The type of upload procedure.
proc_object_id	UNSIGNED BIGINT	The object ID of the stored procedure to use for the publication.

Constraints on underlying system table

```
PRIMARY KEY (pub_object_id, table_object_id, type)
```

```
FOREIGN KEY (pub_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
```

```
FOREIGN KEY (table_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
```

```
FOREIGN KEY (proc_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
```

SYSSYNCSCRIPTS consolidated view

Each row in the SYSSYNCSCRIPTS view identifies a stored procedure for scripted upload. This view is almost identical to the SYSSYNCSCRIPT system view, except that the values are in human-readable format, as opposed to raw data.

```
ALTER VIEW "SYS"."SYSSYNCSCRIPTS"
  as select p.publication_name,
    t.table_name,
    case s.type
    when 0 then 'upload insert'
    when 1 then 'upload delete'
    when 2 then 'upload update'
    else 'unknown'
    end as type,
    c.proc_name
    from SYS.ISYSSYNCSCRIPT as s
      join SYS.ISYSPUBLICATION as p on p.object_id = s.pub_object_id
      join SYS.ISYSTAB as t on t.object_id = s.table_object_id
      join SYS.ISYSPROCEDURE as c on c.object_id = s.proc_object_id
```

SYSSYNCSUBSCRIPTIONS consolidated view

The SYSSYNCSUBSCRIPTIONS view contains the synchronization settings associated with synchronization subscriptions.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

Values in the server_connect and option columns are hidden from users that do not have the SELECT ANY TABLE system privilege.

```
ALTER VIEW "SYS"."SYSSYNCSUBSCRIPTIONS"
  as select s.sync_id,
    p.publication_name,
    s.progress,
    s.site_name,
    SYS.HIDE_FROM_NON_DBA(s."option") as "option",
    SYS.HIDE_FROM_NON_DBA(s.server_connect) as server_connect,
    s.server_conn_type,
    s.last_download_time,
    s.last_upload_time,
    s.created,
    s.log_sent,
    s.generation_number,
    s.extended_state
    from SYS.ISYSSYNC as s join SYS.ISYSPUBLICATION as p
on(p.publication_id = s.publication_id)
    where s.publication_id is not null and
    s.site_name is not null and exists
    (select 1 from SYS.SYSSYNCUSERS as u
      where s.site_name = u.site_name)
```

SYSSYNCUSERS consolidated view

A view of synchronization settings associated with synchronization users.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

Values in the server_connect and option columns are hidden from users that do not have the SELECT ANY TABLE system privilege.

```
ALTER VIEW "SYS"."SYSSYNCUSERS"
  as select ISYSSYNC.sync_id,
    ISYSSYNC.site_name,
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC."option") as "option",
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC.server_connect) as
server_connect,
    ISYSSYNC.server_conn_type
    from SYS.ISYSSYNC where
    ISYSSYNC.publication_id is null
```

SYSTAB system view

Each row of the SYSTAB system view describes one table or view in the database. Additional information for views can be found in the SYSVIEW system view. The underlying system table for this view is ISYSTAB.

Column name	Data type	Description
table_id	UNSIGNED INT	Each table is assigned a unique number (the table number).
dbspace_id	SMALLINT	A value indicating which dbspace contains the table.
count	UNSIGNED BIGINT	The number of rows in the table or materialized view. This value is updated during each successful checkpoint. This number is used to optimize database access. The count is always 0 for a non-materialized view or remote table.
creator	UNSIGNED INT	The user number of the owner of the table or view.
table_page_count	INTEGER	The total number of main pages used by the underlying table.
ext_page_count	INTEGER	The total number of extension pages used by the underlying table.
commit_action	INTEGER	For global temporary tables, 0 indicates that the ON COMMIT PRESERVE ROWS clause was specified when the table was created, 1 indicates that the ON COMMIT DELETE ROWS clause was specified when the table was created (the default behavior for temporary tables), and 3 indicates that the NOT TRANSACTIONAL clause was specified when the table was created. For non-temporary tables, commit_action is always 0.

Column name	Data type	Description
share_type	INTEGER	For global temporary tables, 4 indicates that the SHARE BY ALL clause was specified when the table was created, and 5 indicates that the SHARE BY ALL clause was *not* specified when the table was created. For non-temporary tables, share_type is always 5 because the SHARE BY ALL clause cannot be specified when creating non-temporary tables.
object_id	UNSIGNED BIGINT	The object ID of the table.
last_modified_at	TIMESTAMP	The local time at which the data in the table was last modified. This column is only updated at checkpoint time.
table_name	CHAR(128)	The name of the table or view. One creator cannot have two tables or views with the same name.
table_type	TINYINT	The type of table or view. Values include: • **1 –** Base table • **2 –** Materialized view • **3 –** Global temporary table • **4 –** Local temporary table • **5 –** Text index base table • **6 –** Text index global temporary table • **21 –** View
replicate	CHAR(1)	This value is for internal use only.
server_type	TINYINT	The location of the data for the underlying table. Values include: • **1 –** Local server • **3 –** Remote server

Column name	Data type	Description
tab_page_list	LONG VARBIT	For internal use only. The set of pages that contain information for the table, expressed as a bitmap.
ext_page_list	LONG VARBIT	For internal use only. The set of pages that contain row extensions and large object (LOB) pages for the table, expressed as a bitmap.
pct_free	UNSIGNED INT	The PCT_FREE specification for the table, if one has been specified; otherwise, NULL.
clustered_index_id	UNSIGNED INT	The ID of the clustered index for the table. If none of the indexes are clustered, then this field is NULL.
encrypted	CHAR(1)	Whether the table or materialized view is encrypted.
last_modified_tsn	UNSIGNED BIGINT	A sequence number assigned to the transaction that modified the table. This column is only updated at checkpoint time.
current_schema	UNSIGNED INT	The current schema version of the table.
file_id	SMALLINT	DEPRECATED. This column is present in SYSVIEW, but not in the underlying system table ISYSTAB. The contents of this column is the same as dbspace_id and is provided for compatibility. Use dbspace_id instead.
table_type_str	CHAR(13)	Readable value for table_type. Values include: • **BASE –** Base table • **MAT VIEW –** Materialized view • **GBL TEMP –** Global temporary table • **VIEW –** View

Column name	Data type	Description
last_modified_at_utc	TIMESTAMP WITH TIME ZONE	The UTC time at which the data in the table was last modified. This column is only updated at checkpoint time.

Constraints on underlying system table

```
FOREIGN KEY (dbspace_id) REFERENCES SYS.ISYSDBSPACE (dbspace_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id)
```

```
PRIMARY KEY (table_id)
```

```
FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
```

```
UNIQUE INDEX (table_name, creator)
```

SYSTABAUTH consolidated view

The SYSTABAUTH view contains information from the SYSTABLEPERM system view, but in a more readable format.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSTABAUTH"( grantor,
  grantee,screator,stname,tcreator,ttname,
  selectauth,insertauth,deleteauth,
  updateauth,updatecols,alterauth,referenceauth,
  loadauth,truncateauth )
  as select u1.user_name,u2.user_name,u3.user_name,tab1.table_name,
    u4.user_name,tab2.table_name,tp.selectauth,tp.insertauth,
    tp.deleteauth,tp.updateauth,tp.updatecols,tp.alterauth,
    tp.referenceauth,tp.loadauth,tp.truncateauth
    from SYS.ISYSTABLEPERM as tp
      join SYS.ISYSUSER as u1 on u1.user_id = tp.grantor
      join SYS.ISYSUSER as u2 on u2.user_id = tp.grantee
      join SYS.ISYSTAB as tab1 on tab1.table_id = tp.stable_id
      join SYS.ISYSUSER as u3 on u3.user_id = tab1.creator
      join SYS.ISYSTAB as tab2 on tab2.table_id = tp.stable_id
      join SYS.ISYSUSER as u4 on u4.user_id = tab2.creator
```

SYSTABCOL system view

The SYSTABCOL system view contains one row for each column of each table and view in the database. The underlying system table for this view is ISYSTABCOL.

Column name	Data type	Description
table_id	UNSIGNED INT	The object ID of the table or view to which the column belongs.

Column name	Data type	Description
column_id	UNSIGNED INT	The ID of the column. For each table, column numbering starts at 1. The column_id value determines the order of columns in the result set when SELECT * is used. It also determines the column order for an INSERT statement when a list of column names is not provided.
domain_id	SMALLINT	The data type for the column, indicated by a data type number listed in the SYSDOMAIN system view.
nulls	CHAR(1)	Indicates whether NULL values are allowed in the column.
width	BIGINT	The length of a string column, the precision of numeric columns, or the number of bytes of storage for any other data type.
scale	SMALLINT	The number of digits after the decimal point for NUMERIC or DECIMAL data type columns. For string columns, a value of 1 indicates character-length semantics and 0 indicates byte-length semantics.
object_id	UNSIGNED BIGINT	The object ID of the table column.
max_identity	BIGINT	The largest value of the column, if it is an AUTOINCREMENT, IDENTITY, or GLOBAL AUTOINCREMENT column.
column_name	CHAR(128)	The name of the column.
"default"	LONG VARCHAR	The default value for the column. This value, if specified, is only used when an INSERT statement does not specify a value for the column.

Column name	Data type	Description
user_type	SMALLINT	The data type, if the column is defined using a user-defined data type.
column_type	CHAR(1)	The type of column (C=computed column, and R=other columns).
compressed	TINYINT	Whether this column is stored in a compressed format.
collect_stats	TINYINT	Whether the system automatically collects and updates statistics on this column.
inline_max	SMALLINT	The maximum number of bytes of a BLOB to store in a row. A NULL value indicates that either the default value has been applied, or that the column is not a character or binary type. A non-NULL inline_max value corresponds to the INLINE value specified for the column using the CREATE TABLE or ALTER TABLE statement.
inline_long	SMALLINT	The number of duplicate bytes of a BLOB to store in a row if the BLOB size exceeds the inline_max value. A NULL value indicates that either the default value has been applied, or that the column is not a character or binary type. A non-NULL inline_long value corresponds to the PREFIX value specified for the column using the CREATE TABLE or ALTER TABLE statement.

Column name	Data type	Description
lob_index	TINYINT	Whether to build indexes on BLOB values in the column that exceed an internal threshold size (approximately eight database pages). A NULL value indicates either that the default is applied, or that the column is not BLOB type. A value of 1 indicates that indexes will be built. A value of 0 indicates that no indexes will be built. A non-NULL lob_index value corresponds to whether INDEX or NO INDEX was specified for the column using the CREATE TABLE or ALTER TABLE statement.
base_type_str	VARCHAR(32,767)	The annotated type string representing the physical type of the column.
nonmaterialized_value	LONG BINARY	Internal use only.
start_schema	UNSIGNED INT	The first version of the table schema in which this column exists.

Constraints on underlying system table

```
PRIMARY KEY (table_id, column_id)
```

```
FOREIGN KEY (table_id) REFERENCES SYS.ISYSTAB (table_id)
```

```
FOREIGN KEY (domain_id) REFERENCES SYS.ISYSDOMAIN (domain_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
FOREIGN KEY (user_type) REFERENCES SYS.ISYSUSERTYPE (type_id)
```

SYSTABLE compatibility view (deprecated)

The SYSTABLE view is provided for compatibility with older versions of the software that offered a SYSTABLE system table. However, the SYSTABLE system table has been replaced by the ISYSTAB system table, and its corresponding SYSTAB system view, which you should use instead.

Each row of SYSTABLE view describes one table in the database.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSTABLE"
  as select b.table_id,
    b.file_id,
    b.count,
    0 as first_page,
    b.commit_action as last_page,
    COALESCE(ph.root,0) as primary_root,
    b.creator,
    0 as first_ext_page,
    0 as last_ext_page,
    b.table_page_count,
    b.ext_page_count,
    b.object_id,
    b.table_name,
    b.table_type_str as table_type,
    v.view_def,
    r.remarks,
    b.replicate,
    p.existing_obj,
    p.remote_location,
    'T' as remote_objtype,
    p.srvid,
    case b.server_type
    when 1 then 'SA'
    when 2 then 'IQ'
    when 3 then 'OMNI'
    else 'INVALID'
    end as server_type,
    10 as primary_hash_limit,
    0 as page_map_start,
    s.source,
    b."encrypted"
    from SYS.SYSTAB as b
      left outer join SYS.ISYSREMARK as r on(b.object_id =
r.object_id)
      left outer join SYS.ISYSSOURCE as s on(b.object_id =
s.object_id)
      left outer join SYS.ISYSVIEW as v on(b.object_id =
v.view_object_id)
      left outer join SYS.ISYSPROXYTAB as p on(b.object_id =
p.table_object_id)
     left outer join(SYS.ISYSIDX as i left outer join SYS.ISYSPHYSIDX
as ph on(i.table_id = ph.table_id
        and i.phys_index_id = ph.phys_index_id)) on(b.table_id =
i.table_id and i.index_category = 1
        and i.index_id = 0)
```

SYSTABLEPERM system view

Privileges granted on tables and views by the GRANT statement are stored in the SYSTABLEPERM system view. Each row in this view corresponds to one table, one user ID granting the privilege (grantor) and one user ID granted the privilege (grantee). The underlying system table for this view is ISYSTABLEPERM.

Column name	Data type	Description
stable_id	UNSIGNED INT	The table number of the table or view to which the privileges apply.
grantee	UNSIGNED INT	The user number of the user ID receiving the privilege.
grantor	UNSIGNED INT	The user number of the user ID granting the privilege.
selectauth	CHAR(1)	Indicates whether SELECT privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
insertauth	CHAR(1)	Indicates whether INSERT privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
deleteauth	CHAR(1)	Indicates whether DELETE privileges has been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
updateauth	CHAR(1)	Indicates whether UPDATE privileges have been granted for all columns in the table. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.

Column name	Data type	Description
updatecols	CHAR(1)	Indicates whether UPDATE privileges have only been granted for some of the columns in the underlying table. If updatecols has the value Y, there will be one or more rows in the SYSCOLPERM system view granting update privileges for the columns.
alterauth	CHAR(1)	Indicates whether ALTER privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
referenceauth	CHAR(1)	Indicates whether REFERENCE privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
loadauth	CHAR(1)	Indicates whether LOAD privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
truncateauth	CHAR(1)	Indicates whether TRUNCATE privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
loadauth	CHAR(1)	Indicates whether LOAD privileges has been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.

Column name	Data type	Description
truncateauth	CHAR(1)	Indicates whether TRUNCATE privileges has been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.

Remarks

There are several types of privileges that can be granted. Each privilege can have one of the following three values.

- **N –** No, the grantee has not been granted this privilege by the grantor.
- **Y –** Yes, the grantee has been given this privilege by the grantor.
- **G –** The grantee has been given this privilege and can grant the same privilege to another user.

Note: The grantee might have been given the privilege for the same table by another grantor. If so, this information would be found in a different row of the SYSTABLEPERM system view.

Constraints on underlying system table

```
PRIMARY KEY (stable_id, grantee, grantor)
```

```
FOREIGN KEY (stable_id) REFERENCES SYS.ISYSTAB (table_id)
```

```
FOREIGN KEY (grantor) REFERENCES SYS.ISYSUSER (user_id)
```

```
FOREIGN KEY (grantee) REFERENCES SYS.ISYSUSER (user_id)
```

SYSTEXTCONFIG system view

Each row in the SYSTEXTCONFIG system view describes one text configuration object, for use with the full text search feature. The underlying system table for this view is ISYSTEXTCONFIG.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The object ID for the text configuration object.
creator	UNSIGNED INT	The creator of the text configuration object.

Column name	Data type	Description
term_breaker	TINYINT	The algorithm used to separate a string into terms or words. Values are 0 for GENERIC and 1 for NGRAM. With GENERIC, any string of one or more alphanumeric characters separated by non-alphanumerics are treated as a term. NGRAM is for approximate matching or for documents that do not use a whitespace to separate terms.
stemmer	TINYINT	For internal use only.
min_term_length	TINYINT	The minimum length, in characters, allowed for a term. Terms that are shorter than min_term_length are ignored. The MINIMUM TERM LENGTH setting is only meaningful for the GENERIC term breaker. For NGRAM text indexes, the setting is ignored.
max_term_length	TINYINT	For GENERIC text indexes, the maximum length, in characters, allowed for a term. Terms that are longer than max_term_length are ignored. For NGRAM text indexes, this is the length of the n-grams into which terms are broken.
collation	CHAR(128)	For internal use only.
text_config_name	CHAR(128)	The name of the text configuration object.
prefilter	LONG VARCHAR	The function and library name for an external prefilter library.
postfilter	LONG VARCHAR	For internal use only.

Column name	Data type	Description
char_stoplist	LONG VARCHAR	Terms to ignore when performing a full text search on CHAR columns. These terms are also omitted from text indexes. This column is used when the text configuration object is created from default_char.
nchar_stoplist	LONG NVARCHAR	Terms to ignore when performing a full text search on NCHAR columns. These terms are also omitted from text indexes. This column is used when the text configuration object is created from default_nchar.
external_term_breaker	LONG VARCHAR	The function and library name for an external term breaker library.

Constraints on underlying system table

```
PRIMARY KEY (object_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id ) MATCH
UNIQUE FULL
```

```
FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
```

```
UNIQUE INDEX (creator, text_config_name)
```

SYSTEXTIDX system view

Each row in the SYSTEXTIDX system view describes one text index. The underlying system table for this view is ISYSTEXTIDX.

Column name	Data type	Description
index_id	UNSIGNED BIGINT	The object ID of the text index in SYSIDX.
sequence	UNSIGNED INT	For internal use only.
status	UNSIGNED INT	For internal use only.
text_config	UNSIGNED BIGINT	The object ID of the text configuration object in SYSTEXTCONFIG.
next_handle	UNSIGNED INT	For internal use only.
last_handle	UNSIGNED INT	For internal use only.

Column name	Data type	Description
deleted_length	UNSIGNED BIGINT	The total size of deleted indexed values in the text index.
pending_length	UNSIGNED BIGINT	The total size of indexed values that will be added to the text index at the next refresh.
refresh_type	TINYINT	The type of refresh. One of: • **1 –** MANUAL • **2 –** AUTO • **3 –** IMMEDIATE
refresh_interval	UNSIGNED INT	The AUTO REFRESH interval, in minutes.
last_refresh	TIMESTAMP	The local time of the last refresh.
last_refresh_utc	TIMESTAMP WITH TIME ZONE	The UTC time of the last refresh.

Constraints on underlying system table

```
PRIMARY KEY (index_id, sequence)
```

```
FOREIGN KEY (index_id) REFERENCES SYS.ISYSOBJECT (object_id)
```

```
FOREIGN KEY (text_config) REFERENCES SYS.ISYSTEXTCONFIG (object_id)
```

SYSTEXTIDXTAB system view

Each row in the SYSTEXTIDXTAB system view describes a generated table that is part of a text index. The underlying system table for this view is ISYSTEXTIDXTAB.

Column name	Data type	Description
index_id	UNSIGNED BIGINT	For internal use only.
sequence	UNSIGNED INT	For internal use only.
table_type	UNSIGNED INT	For internal use only.
table_id	UNSIGNED INT	For internal use only.

Constraints on underlying system table

```
PRIMARY KEY (index_id, sequence, table_type)
```

```
FOREIGN KEY (index_id, sequence) REFERENCES SYS.ISYSTEXTIDX
(index_id, sequence)
```

```
FOREIGN KEY (table_id) REFERENCES SYS.ISYSTAB (table_id)
```

SYSTRIGGER system view

Each row in the SYSTRIGGER system view describes one trigger in the database. This view also contains triggers that are automatically created for foreign key definitions which have a referential triggered action (such as ON DELETE CASCADE). The underlying system table for this view is ISYSTRIGGER.

Column name	Data type	Description
trigger_id	UNSIGNED INT	A unique number for the trigger in the SYSTRIGGER view.
table_id	UNSIGNED INT	The table ID of the table to which this trigger belongs.
object_id	UNSIGNED BIGINT	The object ID for the trigger in the database.
event	CHAR(1)	The operation that causes the trigger to fire. • **A –** INSERT, DELETE • **B –** INSERT, UPDATE • **C –** UPDATE COLUMNS • **D –** DELETE • **E –** DELETE, UPDATE • **I –** INSERT • **M –** INSERT, DELETE, UPDATE • **U –** UPDATE
trigger_time	CHAR(1)	The time when the trigger fires relative to the event. • **A –** AFTER (row-level trigger) • **B –** BEFORE (row-level trigger) • **I –** INSTEAD OF (row-level trigger) • **K –** INSTEAD OF (statement-level trigger) • **R –** RESOLVE • **S –** AFTER (statement-level trigger)

Column name	Data type	Description
trigger_order	SMALLINT	The order in which are fired when there are multiple triggers of the same type (insert, update, or delete) set to fire at the same time (applies to BEFORE or AFTER triggers only, only).
foreign_table_id	UNSIGNED INT	The ID of the table containing a foreign key definition that has a referential triggered action (such as ON DELETE CASCADE). The foreign_table_id value reflects the value of ISYSIDX.table_id.
foreign_key_id	UNSIGNED INT	The ID of the foreign key for the table referenced by foreign_table_id. The foreign_key_id value reflects the value of ISYSIDX.index_id.
referential_action	CHAR(1)	The action defined by a foreign key. This single-character value corresponds to the action that was specified when the foreign key was created. • **C –** CASCADE • **D –** SET DEFAULT • **N –** SET NULL • **R –** RESTRICT
trigger_name	CHAR(128)	The name of the trigger. One table cannot have two triggers with the same name.
trigger_defn	LONG VARCHAR	The command that was used to create the trigger.
remarks	LONG VARCHAR	Remarks about the trigger. This value is stored in the ISYSREMARK system table.
source	LONG VARCHAR	The SQL source for the trigger. This value is stored in the ISYSSOURCE system table.

Constraints on underlying system table

```
PRIMARY KEY (trigger_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
FOREIGN KEY (table_id) REFERENCES SYS.ISYSTAB (table_id)
```

```
FOREIGN KEY fkey_index (foreign_table_id, foreign_key_id) REFERENCES
SYS.ISYSIDX (table_id, index_id)
```

```
UNIQUE INDEX (table_id, event, trigger_time, trigger_order)
```

```
UNIQUE INDEX (trigger_name, table_id)
```

```
UNIQUE INDEX (table_id, foreign_table_id, foreign_key_id, event)
```

SYSTRIGGERS consolidated view

Each row in the SYSTRIGGERS view describes one trigger in the database. This view also contains triggers that are automatically created for foreign key definitions which have a referential triggered action (such as ON DELETE CASCADE).

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSTRIGGERS"( owner,
  trigname,tname,event,trigtime,trigdefn )
  as select u.user_name,trig.trigger_name,tab.table_name,
    if trig.event = 'I' then 'INSERT'
    else if trig.event = 'U' then 'UPDATE'
      else if trig.event = 'C' then 'UPDATE'
        else if trig.event = 'D' then 'DELETE'
          else if trig.event = 'A' then 'INSERT,DELETE'
            else if trig.event = 'B' then 'INSERT,UPDATE'
              else if trig.event = 'E' then 'DELETE,UPDATE'
                else 'INSERT,DELETE,UPDATE'
                endif
              endif
            endif
          endif
        endif
      endif
    endif,if trig.trigger_time = 'B' or trig.trigger_time = 'P' then
'BEFORE'
    else if trig.trigger_time = 'A' or trig.trigger_time = 'S' then
'AFTER'
      else if trig.trigger_time = 'R' then 'RESOLVE'
        else 'INSTEAD OF'
        endif
      endif
    endif,trig.trigger_defn
    from SYS.ISYSTRIGGER as trig
      join SYS.ISYSTAB as tab on(tab.table_id = trig.table_id)
      join SYS.ISYSUSER as u on u.user_id = tab.creator where
    trig.foreign_table_id is null
```

SYSTYPEMAP system view

The SYSTYPEMAP system view contains the compatibility mapping values for entries in the SYSSQLSERVERTYPE system view. The underlying system table for this view is ISYSTYPEMAP.

Column name	Data type	Description
ss_user_type	SMALLINT	Contains the Adaptive Server Enterprise user type.
sa_domain_id	SMALLINT	Contains the corresponding SAP Sybase IQ domain_id.
sa_user_type	SMALLINT	Contains the corresponding SAP Sybase IQ user type.
nullable	CHAR(1)	Whether the type allows NULL values.

Constraints on underlying system table

```
FOREIGN KEY (sa_domain_id) REFERENCES SYS.ISYSDOMAIN (domain_id)
```

SYSTYPES ASE Compatibility View

systypes contains one row for each system-supplied and user-defined datatype. Domains (defined by rules) and defaults are given, if they exist.

This view is owned by user DBO. You cannot alter the rows that describe system-supplied datatypes.

See also

- *Tables in Each Adaptive Server Enterprise Database* on page 810
- *SYSCOLUMNS ASE Compatibility View* on page 703
- *SYSCOMMENTS ASE Compatibility View* on page 703
- *SYSINDEXES ASE Compatibility View* on page 726
- *SYSIQOBJECTS ASE Compatibility View* on page 738
- *SYSIQVINDEX ASE Compatibility View* on page 742
- *SYSOBJECTS ASE Compatibility View* on page 750
- *SYSUSERS ASE Compatibility View* on page 805

SYSUSER system view

Each row in the SYSUSER system view describes a user in the system. Standalone roles are also stored in this view as well, but only the user_id, object_id, user_name, and user_type columns are meaningful for these roles. The underlying system table for this view is ISYSUSER.

Column name	Data type	Description
user_id	UNSIGNED INT	A unique identifier for the user assigned to the login policy.
object_id	UNSIGNED BIGINT	A unique identifier for the user in the database.
user_name	CHAR(128)	The login name for the user.
password	BINARY(128)	The password for the user. For security, data in this column is visible only to users with the SELECT ANY TABLE system privilege.
login_policy_id	UNSIGNED BIGINT	A unique identifier for the login policy.
expired_password_on_login	TINYINT	A value that indicates if the password for the user expires at the next login.
password_creation_time	TIMESTAMP	The local time that the password was created for the user.
failed_login_attempts	UNSIGNED INT	The number of times that a user can fail to log in before the account is locked.
last_login_time	TIMESTAMP	The local time that the user last logged in.

Column name	Data type	Description
user_type	TINYINT	A value that indicates whether the user is a regular user, or a role, or a user extended as a role. And whether the user, role, or extended role can be altered (mutable) or removed. Possible values: • **1 –** Immutable system role. • **5 –** Mutable system role • **9 –** Immutable and removable system role. • **12 –** Mutable and removable user. • **13 –** Mutable and removable role. • **14 –** Mutable and removable user extended as role.
user_dn	CHAR (1024)	An LDAP Distinguished Name (DN) identifier for the user that is unique within a domain and across domains. The DN is used to authenticate with an LDAP server.
user_dn_cached_at	TIMESTAMP	The time that the user_dn column was last cached. This value is used to determine whether to purge an old DN. Regardless of the database server local time zone, the value is stored in Coordinated Universal Time (UTC).
password_creation_time_utc	TIMESTAMP WITH TIME ZONE	The UTC time that the password was created for the user.
last_login_time_utc	TIMESTAMP WITH TIME ZONE	The UTC time that the user last logged in.
dual_password	BINARY(128)	The first and/or second parts of the dual password for the user. For security, data in this column is visible only to users with the SELECT ANY TABLE system privilege.

Column name	Data type	Description
lock_time	TIMESTAMP	Timestamp at which user was locked due to failed login attempts.

Constraints on underlying system table

```
PRIMARY KEY (user_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
FOREIGN KEY (login_policy_id) REFERENCES SYS.ISYSLOGINPOLICY
(login_policy_id)
```

```
UNIQUE INDEX (user_name)
```

SYSUSERAUTH compatibility view (deprecated)

Each row of the SYSUSERAUTH view describes a user, without exposing their user_id. Instead, each user is identified by their user name. Because this view displays passwords, you must have the SELECT ANY TABLE system privilege to view its data.

The SYSUSERAUTH view is provided for compatibility with older versions of the software. Use the SYSROLEGRANTS consolidated view instead.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

Note: Although the title of this view contains the word auth (for authorities), the security model is based on roles and privileges. The data in the view is therefore compiled using role information from the tables and views mentioned in the view definition.

```
ALTER VIEW "SYS"."SYSUSERAUTH"( name,
  password,resourceauth,dbaauth,scheduleauth,user_group )
  as select
SYSUSERPERM.user_name,SYSUSERPERM.password,SYSUSERPERM.resourceauth
,SYSUSERPERM.dbaauth,SYSUSERPERM.scheduleauth,SYSUSERPERM.user_grou
p
    from SYS.SYSUSERPERM
```

SYSUSERAUTHORITY compatibility view (deprecated)

The SYSUSERAUTHORITY view is provided for compatibility with older versions of the software. Use the SYSROLEGRANTS consolidated view instead.

Each row of SYSUSERAUTHORITY system view describes an authority granted to one user ID.

Note: Although the title of this view contains the word authority, the security model is based on roles and privileges. The data in the view is therefore compiled using role information from the tables and views mentioned in the view definition.

```
ALTER VIEW "SYS"."SYSUSERAUTHORITY" as
  select ISYSROLEGRANT.grantee as user_id,
    sp_auth_sys_role_info.auth
    from SYS.ISYSROLEGRANT
      natural join dbo.sp_auth_sys_role_info()
    where ISYSROLEGRANT.grant_type <> (0x02|0x04) and
    not ISYSROLEGRANT.grantee = any(select
sp_auth_sys_role_info.role_id from dbo.sp_auth_sys_role_info())
union
  select ISYSUSER.user_id,
    cast('GROUP' as varchar(20)) as auth
    from SYS.ISYSUSER
    where ISYSUSER.user_name
in( 'SYS','PUBLIC','diagnostics','SYS_SPATIAL_ADMIN_ROLE','rs_systa
bgroup','SA_DEBUG','dbo' ) union
  select ISYSUSER.user_id,
    cast('GROUP' as varchar(20)) as auth
    from SYS.ISYSUSER
    where ISYSUSER.user_type = (0x02|0x04|0x08) union
  select cast(opt.setting as unsigned integer) as user_id,
    cast('PUBLISH' as varchar(20)) as auth
    from SYS.ISYSOPTION as opt
   where opt."option" like '%db_publisher%' and opt.setting not like
'%-1%'
```

SYSUSERLIST compatibility view (deprecated)

The SYSUSERAUTH view is provided for compatibility with older versions of the software.

Each row of the SYSUSERLIST view describes a user, without exposing their user_id and password. Each user is identified by their user name.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSUSERLIST"( name,
  resourceauth,dbaauth,scheduleauth,user_group )
  as select
SYSUSERPERM.user_name,SYSUSERPERM.resourceauth,SYSUSERPERM.dbaauth,
SYSUSERPERM.scheduleauth,SYSUSERPERM.user_group
    from SYS.SYSUSERPERM
```

SYSUSERMESSAGE system view

Each row in the SYSUSERMESSAGE system view holds a user-defined message for an error condition. The underlying system table for this view is ISYSUSERMESSAGE.

Note: Previous versions of the catalog contained a SYSUSERMESSAGES system table. That table has been renamed to be ISYSUSERMESSAGE (without an 'S'), and is the underlying table for this view.

Column name	Data type	Description
error	INTEGER	A unique identifying number for the error condition.
uid	UNSIGNED INT	The user number that defined the message.
description	VARCHAR(255)	The message corresponding to the error condition.
langid	SMALLINT	Reserved.

Constraints on underlying system table

```
FOREIGN KEY (uid) REFERENCES SYS.ISYSUSER (user_id)
```

```
UNIQUE CONSTRAINT (error, langid)
```

SYSUSEROPTIONS consolidated view

The SYSUSEROPTIONS view contains the option settings that are in effect for each user. If a user has no setting for an option, this view displays the public setting for the option.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSUSEROPTIONS"( user_name,
  "option",setting )
  as select u.user_name,
    o."option",
    isnull((select s.setting
      from SYS.ISYSOPTION as s
      where s.user_id = u.user_id
      and s."option" = o."option"),
    o.setting)
    from SYS.SYSOPTIONS as o,SYS.ISYSUSER as u
    where o.user_name = 'PUBLIC'
```

SYSUSERPERM compatibility view (deprecated)

Each row of the SYSUSERPERM view describes one user ID. You must have the SELECT ANY TABLE system privilege to view data in this view.

This view is deprecated because it only shows the authorities and permissions available in previous versions. You should change your application to use the SYSROLEGRANTS consolidated view.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSUSERPERM"
  as select b.user_id,
    b.object_id,
    b.user_name,
    b.password,
    if AA.resourceauth is not null and AA.resourceauth > 0 then
      'Y' else 'N' endif as resourceauth,
    if AA.dbaauth is not null and AA.dbaauth > 0 then
      'Y' else 'N' endif as dbaauth,
    'N' as scheduleauth,
    if exists(select * from SYS.ISYSOPTION as opt
      where opt."option" like '%db_publisher%' and opt.setting not
like '%-1'
      and b.user_id = cast(opt.setting as integer)) then
      'Y' else 'N' endif as publishauth,
    if AA.remotedbaauth is not null and AA.remotedbaauth > 0 then
      'Y' else 'N' endif as remotedbaauth,
    if b.user_type = (0x02|0x04|0x08) or b.user_name
in( 'SYS','PUBLIC','diagnostics','SYS_SPATIAL_ADMIN_ROLE','rs_systa
bgroup','SA_DEBUG','dbo' ) then
      'Y' else 'N' endif as user_group,
    r.remarks
    from SYS.ISYSUSER as b
      left outer join SYS.ISYSREMARK as r on(b.object_id =
r.object_id)
      left outer join(select sum(if sp_auth_sys_role_info.auth =
'RESOURCE' then 1 else 0 endif) as resourceauth,
       sum(if sp_auth_sys_role_info.auth = 'DBA' then 1 else 0 endif)
as dbaauth,
       sum(if sp_auth_sys_role_info.auth = 'REMOTE DBA' then 1 else 0
endif) as remotedbaauth,
        ISYSROLEGRANT.grantee
        from SYS.ISYSROLEGRANT natural join
dbo.sp_auth_sys_role_info()
        where ISYSROLEGRANT.grant_type <> (0x02|0x04)
        and sp_auth_sys_role_info.auth in( 'DBA','RESOURCE','REMOTE
DBA' )
        group by ISYSROLEGRANT.grantee) as AA
      on(AA.grantee = b.user_id)
```

SYSUSERPERMS compatibility view (deprecated)

This view is deprecated because it only shows the authorities and permissions available in previous versions. You should change your application to use the SYSROLEGRANTS consolidated view.

Each row of the SYSUSERPERMS view describes one user ID. However, password information is not included. All users are allowed to read from this view.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSUSERPERMS"
  as select
SYSUSERPERM.user_id,SYSUSERPERM.user_name,SYSUSERPERM.resourceauth,
SYSUSERPERM.dbaauth,
SYSUSERPERM.scheduleauth,SYSUSERPERM.user_group,SYSUSERPERM.publish
auth,SYSUSERPERM.remotedbaauth,SYSUSERPERM.remarks
    from SYS.SYSUSERPERM
```

SYSUSERTYPE system view

Each row in the SYSUSERTYPE system view holds a description of a user-defined data type. The underlying system table for this view is ISYSUSERTYPE.

Column name	Data type	Description
type_id	SMALLINT	A unique identifying number for the user-defined data type.
creator	UNSIGNED INT	The user number of the owner of the data type.
domain_id	SMALLINT	The data type on which this user defined data type is based, indicated by a data type number listed in the SYSDOMAIN system view.
nulls	CHAR(1)	Whether the user-defined data type allows nulls. Possible values are Y, N, or U. A value of U indicates that nullability is unspecified.
width	BIGINT	The length of a string column, the precision of a numeric column, or the number of bytes of storage for any other data type.
scale	SMALLINT	The number of digits after the decimal point for numeric data type columns, and zero for all other data types.
type_name	CHAR(128)	The name for the data type.
"default"	LONG VARCHAR	The default value for the data type.
"check"	LONG VARCHAR	The CHECK condition for the data type.
base_type_str	VARCHAR(32767)	The annotated type string representing the physical type of the user type.

Constraints on underlying system table

```
PRIMARY KEY (type_id)
```

```
FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
```

```
FOREIGN KEY (domain_id) REFERENCES SYS.ISYSDOMAIN (domain_id)
```

```
UNIQUE CONSTRAINT (type_name)
```

SYSUSERS ASE Compatibility View

sysusers contains one row for each user allowed in the database, and one row for each group or roles.

This view is owned by user DBO.

See also

- *Tables in Each Adaptive Server Enterprise Database* on page 810
- *SYSCOLUMNS ASE Compatibility View* on page 703
- *SYSCOMMENTS ASE Compatibility View* on page 703
- *SYSINDEXES ASE Compatibility View* on page 726
- *SYSIQOBJECTS ASE Compatibility View* on page 738
- *SYSIQVINDEX ASE Compatibility View* on page 742
- *SYSOBJECTS ASE Compatibility View* on page 750
- *SYSTYPES ASE Compatibility View* on page 797

SYSVIEW System View

Each row in the SYSVIEW system view describes a view in the database.

You can find additional information about views in the `SYSTAB` system view. The underlying system table for this view is `ISYSVIEW`.

You can also use the **sa_materialized_view_info** system procedure for a readable format of the information for materialized views. Materialized views are only supported for SQL Anywhere tables in the IQ catalog store.

Column Name	Data Type	Description
view_object_id	UNSIGNED BIGINT	The object ID of the view.
view_def	LONG VARCHAR	The definition (query specification) of the view.
mv_build_type	TINYINT	Currently unused.

Column Name	Data Type	Description
mv_refresh_type	TINYINT	The refresh type defined for the view. Possible values are IMMEDIATE (1) and MANUAL (2).
mv_use_in_optimization	TINYINT	Whether the materialized view can be used during query optimization (0=cannot be used in optimization, 1=can be used in optimization)
mv_last_refreshed_at	TIMESTAMP	Indicates the local date and time that the materialized view was last refreshed.
mv_known_stale_at	TIMESTAMP	The local time at which the materialized view became stale. This value corresponds to the time at which one of the underlying base tables was detected as having changed. A value of 0 indicates that the view is either fresh, or that it has become stale but the database server has not marked it as such because the view has not been used since it became stale. Use the sa_materialized_view_info system procedure to determine the status of a materialized view.
mv_last_refreshed_tsn	UNSIGNED BIGINT	The sequence number assigned to the transaction that refreshed the materialized view.
mv_last_refreshed_at_utc	TIMESTAMP WITH TIME ZONE	Indicates the UTC date and time that the materialized view was last refreshed.

Column Name	Data Type	Description
mv_known_stale_at_utc	TIMESTAMP WITH TIME ZONE	The UTC time at which the materialized view became stale. This value corresponds to the time at which one of the underlying base tables was detected as having changed. A value of 0 indicates that the view is either fresh, or that it has become stale but the database server has not marked it as such because the view has not been used since it became stale. Use the sa_materialized_view_info system procedure to determine the status of a materialized view. This column contains 0 when mv_last_refreshed_at is 0 and NULL when mv_last_refreshed_at is NULL.

Constraints on Underlying System Table

```
PRIMARY KEY (view_object_id)
```

```
FOREIGN KEY (view_object_id) references SYS.ISYSOBJECT (object_id)
MATCH UNIQUE FULL
```

SYSVIEWS consolidated view

Each row of the SYSVIEWS view describes one view, including its view definition.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSVIEWS"( vcreator,
  viewname,viewtext )
  as select u.user_name,t.table_name,v.view_def
    from SYS.ISYSTAB as t
      join SYS.ISYSVIEW as v on(t.object_id = v.view_object_id)
      join SYS.ISYSUSER as u on(u.user_id = t.creator)
```

SYSWEBSERVICE system view

Each row in the SYSWEBSERVICE system view holds a description of a web service. The underlying system table for this view is ISYSWEBSERVICE.

Column name	Data type	Description
service_id	UNSIGNED INT	A unique identifying number for the web service.
object_id	UNSIGNED BIGINT	The ID of the webservice.
service_name	CHAR(128)	The name assigned to the web service.
service_type	VARCHAR(40)	The type of the service; for example, RAW, HTTP, XML, SOAP, or DISH.
auth_required	CHAR(1)	Whether all requests must contain a valid user name and password.
secure_required	CHAR(1)	Whether insecure connections, such as HTTP, are to be accepted, or only secure connections, such as HTTPS.
url_path	CHAR(1)	Controls the interpretation of URLs.
user_id	UNSIGNED INT	If authentication is enabled, identifies the user, or group of users, that have permission to use the service. If authentication is disabled, specifies the account to use when processing requests.
parameter	LONG VARCHAR	A prefix that identifies the SOAP services to be included in a DISH service.
statement	LONG VARCHAR	A SQL statement that is always executed in response to a request. If NULL, arbitrary statements contained in each request are executed instead. Ignored for services of type DISH.

Column name	Data type	Description
remarks	LONG VARCHAR	Remarks about the webservice. This value is stored in the ISYS-REMARK system table.
enabled	CHAR(1)	Indicates whether the web serv-ice is currently enabled or disa-bled (see CREATE SERVICE).

Constraints on underlying system table

```
PRIMARY KEY (service_id)
```

```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH
UNIQUE FULL
```

```
UNIQUE CONSTRAINT (service_name)
```

Transact-SQL Compatibility Views

Adaptive Server Enterprise and SAP Sybase IQ have different system catalogs, reflecting the different uses for the two products.

In Adaptive Server Enterprise, there is a single master database containing a set of system tables holding information that applies to all databases on the server. Many databases may exist within the master database, and each has additional system tables associated with it.

In SAP Sybase IQ, each database exists independently, and contains its own system tables. There is no master database that contains system information on a collection of databases. Each server may run several databases at a time, dynamically loading and unloading each database as needed.

The Adaptive Server Enterprise and SAP Sybase IQ system catalogs are different. The Adaptive Server Enterprise system tables and views are owned by the special user `dbo`, and exist partly in the master database, partly in the **sybsecurity** database, and partly in each individual database; the SAP Sybase IQ system tables and views are owned by the special user `SYS` and exist separately in each database.

To assist in preparing compatible applications, SAP Sybase IQ provides a set of views owned by the special user dbo, which correspond to the Adaptive Server Enterprise system tables and views. Where architectural differences make the contents of a particular Adaptive Server Enterprise table or view meaningless in a SAP Sybase IQ context, the view is empty, containing only the column names and data types.

These topics list the Adaptive Server Enterprise system tables and their implementation in the SAP Sybase IQ system catalog. The owner of all tables is `dbo` in each DBMS.

See also

- *ASE T-SQL Compatibility Views* on page 694

Tables in Each Adaptive Server Enterprise Database

Not all Adaptive Server Enterprise system tables are implemented in the SAP Sybase IQ system catalog.

Table 152. Tables in each Adaptive Server Enterprise database

Table Name	Description	Data?	Supported by SAP Sybase IQ?
sysalternates	One row for each user mapped to a database user	No	No
syscolumns	One row for each column in a table or view, and for each parameter in a procedure. In SAP Sybase IQ, use the owner name dbo when querying, i.e. dbo.syscolumns.	Yes	Yes
syscomments	One or more rows for each view, rule, default, and procedure, giving SQL definition statement.	Yes	Yes
sysconstraints	One row for each referential and check constraint associated with a table or column.	No	No
sysdepends	One row for each procedure, view, or table that is referenced by a procedure, view.	No	No
sysindexes	One row for each clustered or nonclustered index, and one row for each table with no indexes, and an additional row for each table containing text or image data. In SAP Sybase IQ, use the owner name dbo when querying, i.e. dbo.sysindexes.	Yes	Yes
sysiqobjects	One row for each system table, user table, view, procedure, trigger, event, constraint, domain (sysdomain), domain (sysusertype), column, and index.	Yes	Yes
sysiqvindex	One row for each non-fp iq index.	Yes	Yes
syskeys	One row for each primary, foreign, or common key; set by user (not maintained by Adaptive Server Enterprise).	No	No
syslogs	Transaction log.	No	No

Table Name	Description	Data?	Supported by SAP Sybase IQ?
sysobjects	One row for each table, view, procedure, rule, default, log, and (in tempdb only) temporary object.	Contains compatible data only	Yes
sysprocedures	One row for each view, rule, default, and procedure, giving internal definition.	No	No
sysprotects	User permissions information.	No	No
sysreferences	One row for each referential integrity constraint declared on a table or column.	No	No
sysroles	Maps server-wide roles to local database groups.	No	No
syssegments	One row for each segment (named collection of disk pieces).	No	No
systhresholds	One row for each threshold defined for the database.	No	No
systypes	One row for each system-supplied and user-defined data type.	Yes	Yes
sysusers	One row for each user allowed in the database.	Yes	Yes

See also

- *SYSCOLUMNS ASE Compatibility View* on page 703
- *SYSCOMMENTS ASE Compatibility View* on page 703
- *SYSINDEXES ASE Compatibility View* on page 726
- *SYSIQOBJECTS ASE Compatibility View* on page 738
- *SYSIQVINDEX ASE Compatibility View* on page 742
- *SYSOBJECTS ASE Compatibility View* on page 750
- *SYSTYPES ASE Compatibility View* on page 797
- *SYSUSERS ASE Compatibility View* on page 805

Tables in the Adaptive Server Enterprise Master Database

Not all Adaptive Server Enterprise master database tables are implemented in the SAP Sybase IQ system catalog.

Table 153. ASE master database tables

Table Name	Description	Data?	Supported by SAP Sybase IQ?
syscharsets	One row for each character set or sort order	No	No
sysconfigures	One row for each configuration parameter that can be set by a user	No	No
syscurconfigs	Information about configuration parameters currently being used by the server	No	No
sysdatabases	One row for each database on the server	No	No
sysdevices	One row for each tape dump device, disk dump device, disk for databases, and disk partition for databases	No	No
sysengines	One row for each server currently online	No	No
syslanguages	One row for each language (except U.S. English) known to the server	No	No
syslocks	Information about active locks	No	No
sysloginroles	One row for each server login that possesses a system-defined role	No	No
syslogins	One row for each valid user account	Yes	Yes
sysmessages	One row for each system error or warning	No	No
sysprocesses	Information about server processes	No	No
sysremotelogins	One row for each remote user	No	No
syssrvroles	One row for each server-wide role	No	No
sysservers	One row for each remote server	No	No
sysusages	One row for each disk piece allocated to a database	No	No

Tables in the Adaptive Server Enterprise Sybsecurity Database

No Adaptive Server Enterprise sybsecurity database tables are implemented in the SAP Sybase IQ system catalog.

Table 154. ASE sybsecurity database tables

Table Name	Description	Data?	Supported by SAP Sybase IQ?
sysaudits	One row for each audit record	No	No
sysauditoptions	One row for each global audit option	No	No

Compatibility with Other Sybase Databases

Use the topics in this section to simplify migration to SAP Sybase IQ from other SAP Sybase databases, and to serve as a guide for creating SAP Sybase IQ applications that are compatible with Adaptive Server Enterprise or SQL Anywhere.

Compatibility features are addressed in each new version of SAP Sybase IQ. This sectiion compares SAP Sybase IQ with Adaptive Server Enterprise, and SQL Anywhere.

About SQL Anywhere

SAP Sybase IQ is an extension of SQL Anywhere.

In most cases, SQL syntax, functions, options, utilities, procedures, and other features are common to both products. There are, however, important differences. Do not assume that all features described in SQL Anywhere documentation are supported for SAP Sybase IQ. Use the SAP Sybase IQ documentation.

An Overview of Transact-SQL Support

SAP Sybase IQ, like SQL Anywhere, supports a large subset of *Transact-SQL*, which is the dialect of SQL supported by Sybase Adaptive Server Enterprise.

The goal of Transact-SQL support in SAP Sybase IQ is to provide application portability. Many applications, stored procedures, and batch files can be written for use with both Adaptive Server Enterprise and SAP Sybase IQ databases.

The aim is to write applications to work with both Adaptive Server Enterprise and SAP Sybase IQ. Existing Adaptive Server Enterprise applications generally require some changes to run on SQL Anywhere or SAP Sybase IQ databases.

Transact-SQL support in SAP Sybase IQ takes the following form:

- Most SQL statements are compatible between SAP Sybase IQ and Adaptive Server Enterprise.
- For some statements, particularly in the procedure language used in procedures and batches, a separate Transact-SQL statement is supported along with the syntax supported in earlier versions of SAP Sybase IQ. For these statements, SQL Anywhere and SAP Sybase IQ support two dialects of SQL. In this appendix, we name those dialects Transact-SQL and Watcom-SQL.
- A procedure or batch is executed in either the Transact-SQL or Watcom-SQL dialect. You must use control statements from one dialect only throughout the batch or procedure. For example, each dialect has different flow control statements.

SAP Sybase IQ supports a high percentage of Transact-SQL language elements, functions, and statements for working with existing data.

Further, SAP Sybase IQ supports a very high percentage of the Transact-SQL stored procedure language (**CREATE PROCEDURE** syntax, control statements, and so on), and many, but not all, aspects of Transact-SQL data definition language statements.

There are design differences in the architectural and configuration facilities supported by each product. Device management, user management, and maintenance tasks such as backups tend to be system-specific. Even here, however, SAP Sybase IQ provides Transact-SQL system tables as views, where the tables that are not meaningful in SAP Sybase IQ have no rows. Also, SAP Sybase IQ provides a set of system procedures for some of the more common administrative tasks.

Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ Architectures

Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ are complementary products, with architectures designed to suit their distinct purposes.

SAP Sybase IQ is a high-performance decision-support server designed specifically for data warehousing and analytic processing. SQL Anywhere works well as a workgroup or departmental server requiring little administration, and as a personal database. Adaptive Server Enterprise works well as an enterprise-level server for large databases, with a focus on transaction processing.

This section describes architectural differences among the three products. It also describes the Adaptive Server Enterprise-like tools that SAP Sybase IQ and SQL Anywhere include for compatible database management.

Servers and Databases

The relationship between servers and databases is different in Adaptive Server Enterprise from SAP Sybase IQ and SQL Anywhere.

In Adaptive Server Enterprise, each database exists inside a server, and each server can contain several databases. Users can have login rights to the server, and can connect to the server. They can then connect to any of the databases on that server, provided that they have permissions. System-wide system tables, held in a master database, contain information common to all databases on the server.

In SAP Sybase IQ, there is nothing equivalent to the Adaptive Server Enterprise master database. Instead, each database is an independent entity, containing all of its system tables. Users can have connection rights to a database, rather than to the server. When a user connects, he or she connects to an individual database. There is no system-wide set of system tables maintained at a master database level. Each SAP Sybase IQ database server can dynamically start and stop a database, to which users can maintain independent connections. Sybase strongly recommends that you run only one SAP Sybase IQ database per server.

SQL Anywhere and SAP Sybase IQ provide tools in their Transact-SQL support and Open Server support to allow some tasks to be carried out in a manner similar to Adaptive Server Enterprise. There are differences, however, in exactly how these tools are implemented.

Space Allocation and Device Management

Adaptive Server Enterprise, SQL Anywhere and SAP Sybase IQ use different models for managing devices and allocating disk space initially and later, reflecting the different uses for the products.

For example:

- In Adaptive Server Enterprise, you allocate space in database devices initially using **DISK INIT** and then create a database on one or more database devices. You can add more space using **ALTER DATABASE** or automatically, using thresholds.
- In SAP Sybase IQ, you initially allocate space by listing raw devices in the **CREATE DATABASE** statement. You can add more space manually using **CREATE DBSPACE**. Although you cannot add space automatically, you can create events to warn the DBA before space is actually needed. SAP Sybase IQ can also use file system space. SAP Sybase IQ does not support Transact-SQL **DISK** statements, such as **DISK INIT**, **DISK MIRROR**, **DISK REFIT**, **DISK REINIT**, **DISK REMIRROR**, and **DISK UNMIRROR**.
- SQL Anywhere is similar to SAP Sybase IQ, except that the initial **CREATE DATABASE** statement takes a single file system file instead of a list of raw devices. SQL Anywhere lets you initialize its databases using a command utility named dbinit. SAP Sybase IQ provides an expanded version of this utility called **iqinit** for initializing SAP Sybase IQ databases.

System Tables, Catalog Store, and IQ Main Store

an SAP Sybase IQ database is a joint data store.

The joint store consists of:

- The catalog store includes system tables and stored procedures, and resides in a set of tables that are compatible with SQL Anywhere.
- The permanent IQ main store is the set of SAP Sybase IQ tables. Table data is stored in indexes.
- The temporary store consists of a set of temporary tables which the database server uses for sorting and other temporary processing.

Catalog distinctions and compatibility features include:

- SQL Anywhere and SAP Sybase IQ use a different schema from Adaptive Server Enterprise for the catalog (tables, columns, and so on).

- SQL Anywhere and SAP Sybase IQ provide compatibility views that mimic relevant parts of the Adaptive Server Enterprise system tables, although there are performance implications when using them.
- In Adaptive Server Enterprise, the database owner (user ID `dbo`) owns the catalog objects.
- In SQL Anywhere and SAP Sybase IQ, the system owner (user ID `SYS`) owns the catalog objects.

Note: A `dbo` user ID owns the Adaptive Server Enterprise-compatible system views provided by SAP Sybase IQ.

Data Types

Adaptive Server Enterprise, SQL Anywhere and SAP Sybase IQ treat data types differently.

This section discusses compatibility information for data types.

Note: Data types that are not included in this section are currently supported by all three products.

Bit Data Type

Adaptive Server Enterprise, SQL Anywhere and SAP Sybase IQ support the `BIT` data type, with differences.

The differences are:

- SQL Anywhere permits only 0 or 1.
- Adaptive Server Enterprise and SAP Sybase IQ implicitly convert integral data types to `BIT`. Nonzero values are stored as 1 (TRUE).

Character Data Type Compatibility

Adaptive Server Enterprise, SQL Anywhere and SAP Sybase IQ permit `CHAR` and `VARCHAR` data, but each product treats these types differently.

- SAP Sybase IQ treats all strings as `VARCHAR`, even in a blank-padded database.
- Adaptive Server Enterprise and SAP Sybase IQ differentiate between `CHAR` (fixed-length) and `VARCHAR` (variable-length) data.

  Adaptive Server Enterprise trims trailing blank spaces from `VARCHAR` values. SAP Sybase IQ trims trailing blanks from `VARCHAR` values depending on the form of the data and the operation.

When inserting into `CHAR` or `VARCHAR`:

- SQL Anywhere permits inserting integral data types into `CHAR` or `VARCHAR` (implicit conversion).

- Adaptive Server Enterprise and SAP Sybase IQ require explicit conversion.

The maximum size of a column is determined as follows:

- Adaptive Server Enterprise `CHAR` and `VARCHAR` depend on the logical page size, which can be 2K, 4K, 8K, and 16K. For example:
  - 2K page size allows a column as large as a single row, about 1962 bytes.
  - 4K page size allows a column as large as about 4010 bytes.
- SQL Anywhere supports up to 32K-1 with `CHAR` and `VARCHAR`, and up to 2GB with `LONG VARCHAR`.
- SQL Anywhere supports the name `LONG VARCHAR` and its synonym `TEXT`, while Adaptive Server Enterprise supports only the name `TEXT`, not the name `LONG VARCHAR`.
- SAP Sybase IQ supports `CHAR` and `VARCHAR` up to 32K-1 bytes.

  SAP Sybase IQ also supports up to 512TB (with an SAP Sybase IQ page size of 128KB) and 2PB (with an SAP Sybase IQ page size of 512KB) with `LONG VARCHAR`. For information on the `LONG VARCHAR` data type in SAP Sybase IQ, see *Unstructured Data Analytics*.
- Adaptive Server Enterprise supports `NCHAR`, `NVARCHAR`, `UNICHAR`, `UNIVARCHAR` data types. N is for multibyte character sets; UNI is for single-byte character sets.
- SQL Anywhere and SAP Sybase IQ support Unicode in the `CHAR` and `VARCHAR` data types, rather than as a separate data type.
- For compatibility between SAP Sybase IQ and Adaptive Server Enterprise, always specify a length for character data types.

See also

- *Character Data Types* on page 385

Binary Data Types

Binary data type support differs between Adaptive Server Enterprise, SQL Anywhere and SAP Sybase IQ.

Table 155. Binary data type supported sizes

Data type	Adaptive Server Enterprise	SQL Anywhere	SAP Sybase IQ
`BINARY`	< page size	32KB - 1	255
`VARBINARY`	< page size	32KB - 1	32KB - 1
`LONG BINARY*`	not supported	2GB - 1	512TB (IQ page size 128KB) 2PB (IQ page size 512KB)

Data type	Adaptive Server Enterprise	SQL Anywhere	SAP Sybase IQ
`IMAGE`	2GB	2GB - 1	use `LONG BINARY`*

*For information on the `LONG BINARY` data type in SAP Sybase IQ, see *Unstructured Data Analytics* . This feature requires a separate license.

Adaptive Server Enterprise and SQL Anywhere display binary data differently when projected:

- SAP Sybase IQ supports both Adaptive Server Enterprise and SQL Anywhere display formats.
- If '123' is entered in a `BINARY` field the SQL Anywhere display format is by bytes, as '123'; the Adaptive Server Enterprise display format is by nibbles, as '0x616263'.

See also

- *Binary Data Types* on page 394
- *NEWID Function [Miscellaneous]* on page 280
- *STRTOUUID Function [String]* on page 355
- *UUIDTOSTR Function [String]* on page 369
- *Character Data Types* on page 385

Date, Time, Datetime, and Timestamp Data Types

Although Adaptive Server Enterprise, SQL Anywhere and SAP Sybase IQ all support some form of date and time data, there are some differences.

- SQL Anywhere and SAP Sybase IQ support the 4-byte date and time data types.
- Adaptive Server Enterprise supports an 8-byte `datetime` type, and `timestamp` as a user-defined data type (domain) implemented as binary (8).
- SQL Anywhere and SAP Sybase IQ support an 8-byte `timestamp` type, and an 8-byte `datetime` domain implemented as timestamp. The millisecond precision of the SQL Anywhere/SAP Sybase IQ `datetime` data type differs from that of Adaptive Server Enterprise.

Display formats for dates have different defaults:

- Adaptive Server Enterprise defaults to displaying dates in the format "MMM-DD-YYYY" but can be changed by setting an option.
- SQL Anywhere and SAP Sybase IQ default to the ISO "YYYY-MM-DD" format but can be changed by setting an option.

Time conversions are as follows:

- Adaptive Server Enterprise varies the way it converts time stored in a string to an internal time, depending on whether the fraction part of the second was delimited by a colon or a period.
- SQL Anywhere and SAP Sybase IQ convert times in the same way, regardless of the delimiter.

When you insert a time into a DATETIME column:

- Adaptive Server Enterprise and SAP Sybase IQ default to supplying 1st January 1900.
- SQL Anywhere defaults to supplying the current date.

`TIME` and DATETIME values retrieved from an Adaptive Server Enterprise database change when inserted into a SAP Sybase IQ table with a DATETIME column using **INSERT… LOCATION**. The **INSERT…LOCATION** statement uses Open Client, which has a DATETIME precision of 1/300 of a second.

For example, assume that the following value is stored in a table column in an Adaptive Server Enterprise database:

```
2004-11-08 10:37:22.823
```

When you retrieve and store it in a SAP Sybase IQ table using **INSERT...LOCATION**, the value becomes:

```
2004-11-08 10:37:22.823333
```

Compatibility of Datetime and Time Values from Adaptive Server Enterprise

A `DATETIME` or `TIME` value retrieved from an Adaptive Server Enterprise database using **INSERT...LOCATION** can have a different value due to the datetime precision of Open Client.

For example, the `DATETIME` value in the Adaptive Server Enterprise database is ‘2012-11-08 10:37:22.823’. When you retrieve it and store it in SAP Sybase IQ using **INSERT...LOCATION**, the value becomes ‘2012-11-08 10:37:22.823333’.

BIGTIME and BIGDATETIME Support

SAP Sybase IQ supports the Adaptive Server Enterprise data types `BIGTIME` and `BIGDATETIME` for Component Integration Services (CIS) and **INSERT**...**LOCATION**.

- Component Integration Services with Adaptive Server Enterprise—aseodbc server class proxy tables mapped to Adaptive Server Enterprise tables that contain columns of data type `BIGTIME` and `BIGDATETIME`.

  When you create a proxy table mapped to an Adaptive Server Enterprise table, a `BIGDATETIME` column is mapped to a `TIMESTAMP` column by default, if no mapping is specified. A `BIGTIME` column is mapped to a `TIME` column by default.

  The asejdbc server class does not support the `BIGTIME` and `BIGDATETIME` data types.
- **INSERT**...**LOCATION**—the **INSERT**...**LOCATION** command to load data into SAP Sybase IQ tables from Adaptive Server Enterprise tables that contain columns of data type `BIGTIME` and `BIGDATETIME`.

SAP Sybase IQ inserts the Adaptive Server Enterprise data type `BIGTIME` into the SAP Sybase IQ data type `TIME`.

SAP Sybase IQ inserts the Adaptive Server Enterprise data type `BIGDATETIME` into the SAP Sybase IQ data types `DATETIME`, `DATE`, `TIME`, and `TIMESTAMP`.

Numeric Data Types

Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ have different default precision and scale.

- In Adaptive Server Enterprise, the default is precision 18 scale 0.
- In SQL Anywhere, the default is precision 30 scale 6.
- In SAP Sybase IQ, the default is precision 126 scale 38. Because these defaults are too large for TDS and for some client tools, always specify a precision and scale for SAP Sybase IQ exact numeric types.

Text Data Type

Support for `TEXT` data differs between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

- Adaptive Server Enterprise supports up to 2GB with `LONG VARBINARY` (`LONG BINARY` in SQL Anywhere) and `TEXT`. SQL Anywhere does not support `LONG VARBINARY` as a column type, but uses `LONG BINARY` for the same purpose. SQL Anywhere supports up to 2GB with `LONG BINARY` and `TEXT`.
- SAP Sybase IQ supports up to 32KB - 1 with `VARCHAR`. SAP Sybase IQ also supports up to 512TB (with an IQ page size of 128KB) and 2PB (with an IQ page size of 512KB) with `LONG VARCHAR`. For information on the `LONG VARCHAR` data type in SAP Sybase IQ, see *Unstructured Data Analytics*.

Image Data Type

Support for `IMAGE` data differs between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

- Adaptive Server Enterprise and SQL Anywhere support up to 2GB with `IMAGE`.
- SAP Sybase IQ supports up to 512TB (with an IQ page size of 128KB) and 2PB (with an IQ page size of 512KB) with `LONG BINARY`. For information on the `LONG BINARY` data type in SAP Sybase IQ, see *Unstructured Data Analytics*.

Java Data Types

Adaptive Server Enterprise allows Java data types in the database. SQL Anywhere and SAP Sybase IQ do not.

Data Definition Language

Differences exist between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ in how you create databases and database objects.

Creating a Transact-SQL Compatible Database Using the CREATE DATABASE statement

Use Interactive SQL to create a Transact-SQL compatible database.
Type the following statement, for example, in Interactive SQL:

```
CREATE DATABASE 'db-name.db' CASE RESPECT BLANK PADDING ON
```

Case-sensitivity

Case-sensitivity in databases refers to the case-sensitivity of data, identifiers, and passwords.

Case-sensitivity of Data

The case-sensitivity of the data is reflected in indexes, in the results of queries, and so on.

You decide the case-sensitivity of SAP Sybase IQ data in comparisons when you create the database. By default, SAP Sybase IQ databases are case-sensitive in comparisons, although data is always held in the case in which you enter it.

Adaptive Server Enterprise sensitivity to case depends on the sort order installed on the Adaptive Server Enterprise system. You can change case-sensitivity for single-byte character sets by reconfiguring the Adaptive Server Enterprise sort order.

Case-sensitivity of Identifiers

Identifiers include table names, column names, user IDs, and so on.

SAP Sybase IQ does not support case-sensitive identifiers. In Adaptive Server Enterprise, the case-sensitivity of identifiers follows the case-sensitivity of the data.

In Adaptive Server Enterprise, user-defined data type names are case-sensitive. In SAP Sybase IQ, they are case-insensitive.

Case-sensitivity of User IDs and Passwords

Case-sensitivity of passwords is treated differently from other identifiers.

In SAP Sybase IQ and SQL Anywhere, all passwords in newly-created databases are case-sensitive, regardless of the case-sensitivity of the database. The default user ID is DBA and the password for this user is lowercase *sql*.

When you rebuild an existing database, SAP Sybase IQ and SQL Anywhere determine the case-sensitivity of the password as follows:

- If the database was originally entered in a case-insensitive database, the password remains case-insensitive.
- If the password was originally entered in a case-sensitive database, uppercase and mixed-case passwords remain case-sensitive. If the password was entered in all lowercase, then the password becomes case-insensitive.
- Changes to both existing passwords and new passwords are case-sensitive.

In SAP® Sybase Adaptive Server Enterprise, the case-sensitivity of user IDs and passwords follows the case-sensitivity of the server.

Ensuring Compatible Object Names

Each database object must have a unique name within a certain name space.

Outside this name space, duplicate names are allowed. Some database objects occupy different name spaces in Adaptive Server Enterprise as compared to SQL Anywhere and SAP Sybase IQ.

Table Name Uniqueness

Table name uniqueness requirements apply within a database:

- For SAP Sybase IQ and SQL Anywhere, table names must be unique within a database for a given owner. For example, both `user1` and `user2` can create a table called `employee`; uniqueness is provided by the fully qualified names, `user1.employee` and `user2.employee`.
- For Adaptive Server Enterprise, table names must be unique within the database and to the owner.

Index Name Uniqueness

Index name uniqueness requirements apply within a table. In all three products, indexes are owned by the owner of the table on which they are created. Index names must be unique on a given table, but any two tables can have an index of the same name, even for the same owner. For example, in all three products, tables `t1` and `t2` can have indexes of the same name, whether they are owned by the same or different users.

Renaming Indexes and Foreign Keys

SAP Sybase IQ allows you to rename explicitly created indexes, foreign key role names of indexes, and foreign keys, using the **ALTER INDEX** statement. SQL Anywhere allows you to rename indexes, foreign key role names, and foreign keys, using the **ALTER INDEX** statement. Adaptive Server Enterprise does not allow you to rename these objects.

Considerations When Using the CREATE TABLE Statement

When creating tables for compatibility, be aware of the following compatibility considerations for NULL treatment, check constraints, referential integrity, default values, identify columns, computed columns, temporary tables, and table location.

NULL in Columns

For compatible treatment of NULL:

- SQL Anywhere and SAP Sybase IQ assume that columns can be null unless NOT NULL is stated in the column definition. You can change this behavior by setting the database option **ALLOW_NULLS_BY_DEFAULT** to the Transact-SQL compatible setting of OFF.
- SQL Anywhere and SAP Sybase IQ assume that BIT columns cannot be NULL.
- Adaptive Server Enterprise assumes that columns cannot be null unless NULL is stated.

Check Constraints

SAP Sybase IQ enforces check constraints on base, global temporary, and local temporary tables, and on user-defined data types. Users can log check integrity constraint violations and specify the number of violations that can occur before a **LOAD** statement rolls back.

SAP Sybase IQ does not allow the creation of a check constraint that it cannot evaluate, such as those composed of user-defined functions, proxy tables, or non-SAP Sybase IQ tables. Constraints that cannot be evaluated are detected the first time the table on which the check constraint is defined is used in a **LOAD**, **INSERT**, or **UPDATE** statement. SAP Sybase IQ does not allow check constraints containing:

- Subqueries
- Expressions specifying a host language parameter, a SQL parameter, or a column as the target for a data value
- Set functions
- Invocations of nondeterministic functions or functions that modify data

Adaptive Server Enterprise and SQL Anywhere enforce **CHECK** constraints. SQL Anywhere allows subqueries in check constraints.

SAP Sybase IQ supports user-defined data types that allow constraints to be encapsulated in the data type definition.

Referential Integrity Constraints

Actions for enforcing integrity are supported as follows:

- SQL Anywhere supports all ANSI actions: SET NULL, CASCADE, DEFAULT, RESTRICT.
- Adaptive Server Enterprise supports two of these actions: SET NULL, DEFAULT.

Note: You can achieve CASCADE in Adaptive Server Enterprise by using triggers instead of referential integrity.

- SAP Sybase IQ supports the RESTRICT action only.
- SAP Sybase IQ does not support NOT NULL FOREIGN KEY.
- SAP Sybase IQ has the restriction that a column cannot be both a candidate key and a foreign key at the same time.

Default Values in a Column

Default value support differs as follows:

- Adaptive Server Enterprise and SQL Anywhere support specifying a default value for a column.
- Only SQL Anywhere supports DEFAULT UTC TIMESTAMP.
- SAP Sybase IQ supports specifying a default value for a column, except for the special values DEFAULT UTC TIMESTAMP and DEFAULT CURRENT UTC TIMESTAMP. SAP Sybase IQ also ignores settings for the `DEFAULT_TIMESTAMP_INCREMENT` database option.

Identity Columns

Identity column support differs as follows:

- SAP Sybase IQ supports `IDENTITY` or `DEFAULT AUTOINCREMENT` as a default value. SAP Sybase IQ supports identity columns of any numeric type with any precision and scale 0, and the column can be NULL. SAP Sybase IQ identity columns must be positive and are limited by the range of the data type. SAP Sybase IQ supports a single identity column per table, and requires database option **IDENTITY_INSERT** set to a table name for explicit inserts and updates. To drop a table with an `IDENTITY` column, you cannot have **IDENTITY_INSERT** set to that table. The table can contain data when adding an identity column. Tables derived using **SELECT INTO** do not have Identity/Autoincrement columns. SAP Sybase IQ views cannot contain `IDENTITY/DEFAULT AUTOINCREMENT` columns.
- SQL Anywhere supports the `AUTOINCREMENT` default value. SQL Anywhere supports identity columns of any numeric type with any allowable scale and precision. The identity column value can be positive, negative, or zero, limited by the range of the data type. SQL Anywhere supports any number of identity columns per table, and does not require identity_insert for explicit inserts, drops, and updates. The table must be empty when adding identity columns. SQL Anywhere identity columns can be altered to be nonidentity columns, and vice versa. You can add or drop `AUTOINCREMENT` columns from SQL Anywhere views.
- Adaptive Server Enterprise supports a single identity column per table. Adaptive Server Enterprise identity columns are restricted to only numeric data type scale 0, maximum precision 38. They must be positive, are limited by the range of the data type, and cannot be null. Adaptive Server Enterprise requires identity_insert for explicit inserts and drops, but not for updates to the identity column. The table can contain data when you add an identity

column. Adaptive Server Enterprise users cannot explicitly set the next value chosen for an identity column. Adaptive Server Enterprise views cannot contain `IDENTITY/ AUTOINCREMENT` columns. When using **SELECT INTO** under certain conditions, Adaptive Server Enterprise allows Identity/Autoincrement columns in the result table if they were in the table being selected from.

Computed Columns

Computed column support differs as follows:

- SQL Anywhere supports computed columns that can be indexed.
- Adaptive Server Enterprise and SAP Sybase IQ do not.

Temporary Tables

You can create a temporary table by placing a pound sign (#) without an owner specification in front of the table name in a **CREATE TABLE** statement. These temporary tables are SAP Sybase IQ-declared temporary tables and are available only in the current connection.

Locating Tables

Physical placement of a table is carried out differently in Adaptive Server Enterprise and SAP Sybase IQ. SAP Sybase IQ supports the **ON** *segment-name* clause, but *segment-name* refers to a SAP Sybase IQ dbspace.

Output for sp_iqstatus procedure

```
SAP Sybase IQ (TM)                Copyright (c) 1992-2013 by Sybase,
Inc.
                                          All rights reserved.
Version:                                  16.0.0.160/120507/D/ELAN/
Sun_x64/OS 5.10/
                                          64bit/2012-05-07 17:36:36
Time Now:                                 2013-05-16 09:53:13.590
Build Time:                               2013-05-07 17:36:36
File Format:                              23 on 03/18/1999
Server mode:                          IQ Multiplex Coordinator Server
Catalog Format:                           2
Stored Procedure Revision:                1
Page Size:                                131072/8192blksz/16bpp
Number of Main DB Files :                 3
Main Store Out Of Space:                  N
Number of Shared Temp DB Files:           0
Shared Temp Store Out Of Space:           N
Number of Local Temp DB Files :           1
Local Temp Store Out Of Space:            N
DB Blocks: 1-640000                       IQ_SYSTEM_MAIN
DB Blocks: 1045440-130101439              iqmain1
DB Blocks: 2090880-2346879                iqmain2
Local Temp Blocks: 1-384000               IQ_SYSTEM_TEMP
Create Time:                              2013-05-08 15:54:15.549
Update Time:                              2013-05-16 09:53:00.077
Local Temp Blocks:  1-1600                IQ_SYSTEM_TEMP
Create Time:                              2013-05-08 15:54:15.549
```

```
Update Time:                              2013-05-16 09:53:00.077
Main IQ Buffers:                          510, 64Mb
Temporary IQ Buffers:                     510, 64Mb
Main IQ Blocks Used:                   157379 of 1126400, 13%=1229Mb,
Max Block#: 2128363
Shared Temporary IQ Blocks Used:         0 of 0, 0%=0Mb, Max Block#: 0
Local Temporary IQ Blocks Used:           81 of 358400, 0%=0Mb, Max
Block#: 81
Main Reserved Blocks Available:           25600 of 25600, 100%=200Mb
Shared Temporary Reserved Blocks Available:  0 of 0, 0%=0Mb
Local Temporary Reserved Blocks Available:   25600 of 25600,
100%=200Mb
IQ Dynamic Memory:                        Current: 178mb, Max: 178mb
Main IQ Buffers:                          Used: 99, Locked: 0
Temporary IQ Buffers:                     Used: 5, Locked: 0
Main IQ I/O:                           I: L60904/P29 O: C5463/D11343/
P9486
                                          D:5450 C:51.3
Temporary IQ I/O:                      I: L12526/P0 O: C165/D319/P157
D:160 C:100.0
Other Versions:                           6 = 0Mb
Active Txn Versions:                      0 = C:0Mb/D:0Mb
Last Full Backup ID:                      0
Last Full Backup Time:
Last Backup ID:
Last Backup Type:                         None
Last Backup Time:
DB Updated:                               1
Blocks in next ISF Backup:                0 Blocks: =0Mb
Blocks in next ISI Backup:                0 Blocks: =0Mb
Main Tlvlog Size:                       Pages: 1, Recs: 193, Replays:
0/0
DB File Encryption Status:                OFF
```

Considerations When Using the CREATE DEFAULT, CREATE RULE, and CREATE DOMAIN Statements

SAP Sybase IQ provides an alternative means of incorporating rules.

- Adaptive Server Enterprise supports the Create Default and Create Rule statements to create named defaults.
- SQL Anywhere and SAP Sybase IQ support the **CREATE DOMAIN** statement to achieve the same objective.

Considerations When Using the CREATE TRIGGER Statement

Support for triggers differs between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

- SQL Anywhere supports both row-level and statement-level triggers.
- Adaptive Server Enterprise supports only statement-level triggers.
- SAP Sybase IQ does not support triggers.

Note: A trigger is effectively a stored procedure that is run automatically either immediately before or immediately after an **INSERT**, **UPDATE**, or **DELETE** as part of the same transaction, that can be used to cause a dependent change (for example, to automatically update the name of an employee's manager when the employee is moved to a different department). It can also be used to write an audit trail to identify which modifications made which changes to the database, and at what time.

Considerations When Using the CREATE INDEX Statement

CREATE INDEX syntax differs slightly between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

- Adaptive Server Enterprise and SQL Anywhere support clustered or nonclustered indexes, using the following syntax:

  ```
  CREATE [UNIQUE] [CLUSTERED] INDEX name
  ON table (column,...)
  ON dbspace
  ```

  Adaptive Server Enterprise also allows the **NONCLUSTERED** keyword, but for both products the default is **NONCLUSTERED**.
- Adaptive Server Enterprise **CREATE INDEX** statements work in SQL Anywhere because SQL Anywhere allows, but ignores, the keywords **FILLFACTOR**, **IGNORE_DUP_KEY, SORTED_DATA**, **IGNORE_DUP_ROW**, and **ALLOW_DUP_ROW**.
- SQL Anywhere **CREATE INDEX** syntax supports the **VIRTUAL** keyword for use by its Index Consultant, but not for actual query executions.
- SAP Sybase IQ supports seven specialized index types: **LF**, **HG**, **HNG**, **DATE**, **TIME**, **DTTM**, and **WD**. SAP Sybase IQ also supports a **CMP** index on the relationship between two columns of identical data type, precision, and scale. SAP Sybase IQ defaults to creating an **HG** index unless the index type is specified in the **CREATE INDEX** statement:

  ```
  CREATE [UNIQUE] [type] INDEX name
  ON table (column,...)
  ```

Users, Groups/Roles, and Permissions

There are some differences between the Adaptive Server Enterprise and SQL Anywhere and SAP Sybase IQ models of users and roles/groups.

In Adaptive Server Enterprise, users connect to a server, and each user requires a login ID and password to the server as well as a user ID for each database they want to access on that server.

SQL Anywhere and SAP Sybase IQ users do not require a server login ID. All SQL Anywhere and SAP Sybase IQ users receive a user ID and password for a database.

User Roles/Groups

To allow you to grant permissions to many users at one time, SQL Anywhere and SAP Sybase IQ support user roles while Adaptive Server Enterprise supports user groups. Though basically roles are groups are equivalent, there are some behavioral differences:

- Adaptive Server Enterprise allows each user to be a member of only one group.

- SQL Anywhere and SAP Sybase IQ allow users to be members of multiple roles , and role hierarchies are allowed.

All three products have a `public` role or group, for defining default permissions. Every user automatically becomes a member of the `public` role or group.

Database Object Permissions

GRANT and **REVOKE** statements for granting permissions on individual database objects are very similar in all three products.

- All three products allow SELECT, INSERT, DELETE, UPDATE, and REFERENCES permissions on database tables and views, and UPDATE permissions on selected columns of database tables. SQL Anywhere and SAP Sybase IQ also allow LOAD and TRUNCATE permissions on databsse tables and views.

  For example, the following statement is valid in all three products:

  ```
  GRANT INSERT, DELETE
  ON TITLES
  TO MARY, SALES
  ```

  This statement grants permission to use the **INSERT** and **DELETE** statements on the `TITLES` table to user `MARY` and to the `SALES` role or group.
- All three products allow EXECUTE permissions to be granted on stored procedures.
- Adaptive Server Enterprise also supports **GRANT** and **REVOKE** on additional items:
  - Objects: columns within tables, columns within views, and stored procedures
  - User abilities: **CREATE DATABASE**, **CREATE DEFAULT**, **CREATE PROCEDURE**, **CREATE RULE**, **CREATE TABLE**, **CREATE VIEW**
- SQL Anywhere and SAP Sybase IQ require a user to have the MANAGE ANY OBJECT PRIVILEGE system privilege to grant database objects permissions. (A closely corresponding Adaptive Server Enterprise permission is GRANT ALL, used by a Database Owner.)
- All three products support the **WITH GRANT OPTION** clause, allowing the recipient of permissions to grant them in turn, although SAP Sybase IQ and SQL Anywhere do not permit **WITH GRANT OPTION** to be used on a **GRANT EXECUTE** statement.

Database-wide Permissions

Adaptive Server Enterprise uses a different model for database-wide user permissions.

- SQL Anywhere and SAP Sybase IQ use the SYS_AUTH_DBA_ROLE compatibility role to allow a user full permissions within a database, assuming that the SYS_AUTH_DBA_ROLE compatibility role has not been migrated to a hierarchy of roles to meet customer security requirements..
- The System Administrator in Adaptive Server Enterprise enjoys this permission for all databases on a server.
- The database owner must use the Adaptive Server Enterprise **SETUSER** statement to gain permissions on objects owned by other users.

Adding Users

Adaptive Server Enterprise requires a two-step process to add a user: **sp_addlogin** followed by **sp_adduser**.

SQL Anywhere and SAP Sybase IQ add users in a single step.

SAP Sybase IQ Login Management stored procedures, although not required to add or drop users, allow users with applicable system privileges to add or drop SAP Sybase IQ user accounts. When SAP Sybase IQ User Administration is enabled, these SAP Sybase IQ user accounts allow control user connections and password expirations.

Although SQL Anywhere and SAP Sybase IQ allow Adaptive Server Enterprise system procedures for managing users and groups, the exact syntax and function of these procedures differs in some cases.

See also

- *Adaptive Server Enterprise System Procedures* on page 686

Load Formats

Load format support differs between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

- SAP Sybase IQ handles ASCII, BINARY, and BCP load formats.
- SQL Anywhere, in addition to ASCII and BINARY, also lets you import dBase, Excel, FoxPro, and Lotus file formats.
- Adaptive Server Enterprise handles ASCII and BINARY load formats through BCP.

Note: The syntax of the SAP Sybase IQ and SQL Anywhere **LOAD** statement is based on BCP and designed to offer exactly the same functionality.

Options for Transact-SQL Compatibility

Set SAP Sybase IQ database options using the **SET OPTION** statement.

See the Transact-SQL compatibility options in *Reference: Statements and Options*.

Data Manipulation Language

Query requirements differ between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

General Guidelines for Writing Portable SQL

Even if more than one server supports a given SQL statement, it might be a mistake to assume that default behavior is the same on each system.

General guidelines applicable to writing compatible SQL include:

- When writing SQL for use on more than one database management system, make your SQL statements as explicit as possible.
- Spell out all of the available options, rather than using default behavior.
- Use parentheses to make the order of execution within statements explicit, rather than assuming identical default order of precedence for operators.
- Use the Transact-SQL convention of an @ sign preceding variable names for Adaptive Server Enterprise portability.
- Declare variables and cursors in procedures and batches immediately following a **BEGIN** statement. SAP Sybase IQ requires this, although Adaptive Server Enterprise allows declarations to be made anywhere in a procedure or batch.
- Do not use reserved words from either Adaptive Server Enterprise or SAP Sybase IQ as identifiers in your databases.

Criteria for Writing Compatible Queries

There are two criteria for writing a query that runs on both SAP Sybase IQ and Adaptive Server Enterprise databases.

- The data types, expressions, and search conditions in the query must be compatible.
- The syntax of the **SELECT** statement itself must be compatible.

SAP Sybase IQ supports the following subset of the Transact-SQL **SELECT** statement.

Syntax

```
SELECT [ ALL | DISTINCT ] select-list
…[ INTO #temporary-table-name ]
…[ FROM table-spec,
…     table-spec, … ]
…[ WHERE search-condition ]
…[ GROUP BY column-name, … ]
…[ HAVING search-condition ]
…| [ ORDER BY expression [ ASC | DESC ], … ]      |
   | [ ORDER BY integer [ ASC | DESC ], … ]      |
```

Parameters

```
select-list:
{ table-name.* }…
{ * }…
{ expression }…
{ alias-name = expression }…
{ expression as identifier }…
{ expression as T_string }…
```

```
table-spec:
    [ owner. ]table-name
…     [ [ AS ] correlation-name ]
…
```

```
alias-name:
    identifier | 'string' | “string"
```

The sections that follow provide details on several items to be aware of when writing compatible queries.

See also

- *Variables in Transact-SQL Procedures* on page 843

Subquery Support

SAP Sybase IQ currently provides support for subqueries that is somewhat different from that provided by Adaptive Server Enterprise and SQL Anywhere.

Adaptive Server Enterprise and SQL Anywhere support subqueries in the **ON** clause; SAP Sybase IQ does not currently support this.

UNION in subqueries is supported as follows:

- SQL Anywhere supports **UNION** in both correlated and uncorrelated subqueries.
- SAP Sybase IQ supports **UNION** only in uncorrelated subqueries.
- Adaptive Server Enterprise does not support **UNION** in any subqueries.

SQL Anywhere supports subqueries in many additional places that a scalar value might appear in the grammar. Adaptive Server Enterprise and SAP Sybase IQ follow the ANSI standard as to where subqueries can be specified.

GROUP BY Clause Support

GROUP BY ALL support differs between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

- Adaptive Server Enterprise supports **GROUP BY ALL,** which returns all possible groups including those eliminated by the **WHERE** clause and **HAVING** clause. These have the NULL value for all aggregates.
- SQL Anywhere does not support the **GROUP BY ALL** Transact-SQL extension.

ROLLUP and **CUBE** in the **GROUP BY** clause are supported as follows:

- SAP Sybase IQ and SQL Anywhere support **ROLLUP** and **CUBE** in the **GROUP BY** clause.
- Adaptive Server Enterprise does not currently support **ROLLUP** and **CUBE**.

Adaptive Server Enterprise supports projecting nongrouped columns in the **SELECT** clause. This is known as extended group by semantics and returns a set of values. SAP Sybase IQ supports and SQL Anywhere do not support extended group by semantics. Only SQL Anywhere supports the List() aggregate to return a list of values.

COMPUTE Clause Support

COMPUTE support differs between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

- Adaptive Server Enterprise supports the Transact-SQL **COMPUTE** clause.
- SQL Anywhere and SAP Sybase IQ do not support the Transact-SQL **COMPUTE** clause since it is not in the ANSI standard and this functionality is provided by most third-party front-end tools.

WHERE Clause Support

The WHERE clause differs between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ in support for the Contains() predicate, and treatment of trailing white space in the **Like()** predicate.

- SAP Sybase IQ supports the **Contains()** predicate for word searches in character data (similar to Contains in MS SQL Server and Verity). SAP Sybase IQ uses `WORD` indexes and `TEXT` indexes to optimize these, if possible.
- Adaptive Server Enterprise does not support **Contains()**.

Transact-SQL Outer Joins Support

Supported syntax for outer joins differs between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

- Adaptive Server Enterprise fully supports *= and =* Transact-SQL syntax for outer joins.
- SQL Anywhere and SAP Sybase IQ support Transact-SQL outer joins, but reject some complex Transact-SQL outer joins that are potentially ambiguous.
- SAP Sybase IQ does not support chained (nested) Transact-SQL outer joins. Use ANSI syntax for this type of multiple outer join.

Note: Transact-SQL outer join syntax is deprecated in SQL Anywhere and SAP Sybase IQ.

For detailed information on Transact-SQL outer joins, including ANSI syntax alternatives, see the white paper *Semantics and Compatibility of Transact-SQL Outer Joins*, from *MySybase*. Although written for SQL Anywhere, the information in the document also applies to SAP Sybase IQ.

ANSI Joins Support

Support for ANSI join syntax differs between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

- SAP Sybase IQ does not currently support subqueries in the **ON** clause.
- Adaptive Server Enterprise and SQL Anywhere support subqueries in the **ON** clause.
- A **CONTAINS** condition in the **FROM** clause in queries using ANSI join syntax is supported but may have suboptimal performance. Using Outer Joins for **CONTAINS** in the **FROM**

clause should only be used if the "score" column from each of the **CONTAINS** clauses is required. Otherwise **CONTAINS** should be moved to an **ON** condition or **WHERE** clause.

Full outer join support is as follows:

- SQL Anywhere and SAP Sybase IQ support **FULL OUTER JOIN**.
- Adaptive Server Enterprise does not support **FULL OUTER JOIN**.

Null Comparisons Support

Adaptive Server Enterprise has Transact-SQL extensions that permit predicates to compare the null value.

For example, `{col} = Null` means `{col} Is Null`.

SQL Anywhere and SAP Sybase IQ use ANSI semantics for null comparisons unless the `ANSINULL` option is set to OFF, in which case such comparisons are Adaptive Server Enterprise-compatible.

Note: SQL Anywhere 8.0 and later adds support for the `TDS_EMPTY_STRING_AS_NULL` to offer Adaptive Server Enterprise compatibility in mapping empty strings to the null value.

Zero-length Strings Support

Zero-length strings are treated differently in Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

- Adaptive Server Enterprise treats zero-length strings as the null value.
  Adaptive Server Enterprise users store a single space for blank strings.
- SQL Anywhere and SAP Sybase IQ follow ANSI semantics for zero-length strings, that is, a zero-length string is a real value; it is not null.

HOLDLOCK, SHARED, and FOR BROWSE Support

HOLDLOCK, SHARED, and FOR BROWSE syntax differs between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

- Adaptive Server Enterprise supports HOLDLOCK, SHARED, and FOR BROWSE syntax.
- SQL Anywhere supports HOLDLOCK but does not support SHARED or FOR BROWSE.
- SAP Sybase IQ does not support these keywords.

SQL Function Support

SAP Sybase IQ supports most of the same functions as SQL Anywhere and Adaptive Server Enterprise, with some differences.

- Adaptive Server Enterprise supports the **USING CHARACTERS | USING BYTES** syntax in **PatIndex()**; SQL Anywhere and SAP Sybase IQ do not.

- Adaptive Server Enterprise supports the **Reverse()** function; SQL Anywhere and SAP Sybase IQ do not.
- Adaptive Server Enterprise supports **Len()** as alternative syntax for **Length()**; SQL Anywhere does not support this alternative.
- Adaptive Server Enterprise supports the **Square()** and **Str_Replace()** Microsoft compatibility functions; SQL Anywhere does not.
- SAP Sybase IQ supports **Str_Replace()**.
- Adaptive Server Enterprise and SQL Anywhere support **TSEQUAL()** to compare two timestamps for modification time; SAP Sybase IQ does not support **TSEQUAL()**. (**TSEQUAL** is not relevant in the SAP Sybase IQ table-level versioning model.)
- SAP Sybase IQ supports **ROWID()**; Adaptive Server Enterprise and SQL Anywhere do not.
- SQL Anywhere and SAP Sybase IQ support **Cast()** in addition to Adaptive Server Enterprise's **Convert()** for data type conversions.

  Note: Cast() is the ANSI-compliant name.
- SQL Anywhere and SAP Sybase IQ support **Lcase()** and **Ucase()** as synonyms of **Lower()** and **Upper()**; Adaptive Server Enterprise does not.
- SQL Anywhere and SAP Sybase IQ support the **Locate()** string function; Adaptive Server Enterprise does not.
- SQL Anywhere supports the **IsDate()** and **IsNumeric()** function to test the ability to convert a string to the respective data type; Adaptive Server Enterprise does not. SAP Sybase IQ supports **IsDate()**. You can use **IsNumeric** in SAP Sybase IQ, but CIS functional compensation performance considerations apply.
- SQL Anywhere supports the **NEWID**, **STRTOUID**, and **UUIDTOSTR** functions; Adaptive Server Enterprise does not. These are native functions in SAP Sybase IQ, so CIS functional compensation performance considerations do not apply.

Note: Some SQL functions, including **SOUNDEX** and **DIFFERENCE** string functions, and some date functions operate differently in SAP Sybase IQ and SQL Anywhere. The SAP Sybase IQ database option `ASE_FUNCTION_BEHAVIOR` specifies that output of some of the SAP Sybase IQ data type conversion functions, including **HEXTOINT** and **INTTOHEX**, is consistent with the output of Adaptive Server Enterprise functions.

OLAP Function Support

Currently, Adaptive Server Enterprise does not support OLAP functions. SAP Sybase IQ and SQL Anywhere do.

SAP Sybase IQ currently supports these OLAP functions:

- **Corr()**
- **Covar_Pop()**
- **Covar_Samp()**

- **Cume_Dist**
- **Dense_Rank()**
- **Exp_Weighted_Avg**
- **First_Value**
- **Last_Value**
- **Median**
- **Ntile()**
- **Percent_Rank()**
- **Percentile_Cont()**
- **Percentile_Disc()**
- **Rank()**
- **Regr_Avgx()**
- **Regr_Avgy()**
- **Regr_Intercept()**
- **Regr_R2**
- **Regr_Slope()**
- **Regr_Sxx()**
- **Regr_Sxy**()
- **Regr_Syy()**
- **StdDev()**
- **Stddev_Pop**
- **Stddev_Samp**
- **Var_Pop**
- **Var_Samp**
- **Variance**()
- **Weighted_Avg**

SQL Anywhere supports all of the SAP Sybase IQ OLAP functions.

Currently, Adaptive Server Enterprise does not support OLAP functions.

CIS functional compensation does not support OLAP functions.

Note: Support for OLAP functions is a rapidly evolving area of Sybase product development.

System Function Support

SAP Sybase IQ and SQL Anywhere do not support certain Adaptive Server Enterprise system functions.

These Adaptive Server Enterprise system functions are not supported by SQL Anywhere and SAP Sybase IQ:

- **curunreservedpgs()** – number of pages free on a dbspace.

- **data_pgs()** – number of pages used by a table or index.
- **host_id()** – UNIX pid of the server process.
- **hos_name()** – name of the machine on which the server is running.
- **lct_admin()** – manages the “last chance threshold” for the Transaction manager.
- **reserved_pgs()** – number of pages allocated to a table or index.
- **rowcnt()** – number of rows in the specified table.
- **valid_name()** – whether a name would be a valid name if used, for example, for a table.
- **valid_user()** – returns TRUE if that user has connect permissions.
- **ptn_data_pgs()** – number of data pages in a partition.
- **index_colorder()** – returns the column order in an index.

User-Defined Function Support

User-defined function (UDF) support differs between Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ:

- SQL Anywhere supports UDFs in SQL, Java, and C.
- Adaptive Server Enterprise supports UDFs written only in Java .
- SAP Sybase IQ offers support for external C/C++ and Java UDFs as a licensed option. SAP Sybase IQ offers support for Interactive SQL UDFs via CIS query decomposition, but there are performance implications.

Differences Interpreting Arithmetic Expressions on Dates

SQL Anywhere and SAP Sybase IQ interpret arithmetic expressions on dates as shorthand notation for various date functions. Adaptive Server Enterprise does not.

- Date +/- integer is equivalent to **Dateadd()**.
- Date – date is equivalent to **Datediff()**.
- Date + time creates a timestamp from the two.

SELECT INTO Statement Support

There are differences in the types of tables permitted in SELECT INTO statements in Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

Consider this example statement:

```
select into table1 from table2
```

- Adaptive Server Enterprise permits *table1* to be permanent, temporary or a proxy table. Adaptive Server Enterprise also supports **SELECT INTO EXISTING TABLE**.
- SQL Anywhere and SAP Sybase IQ permit *table1* to be a permanent or a temporary table. A permanent table is created only when you select into *table* and specify more than one column. **SELECT INTO** *#table*, without an owner specification, always creates a temporary table, regardless of the number of columns specified. **SELECT INTO** table with just one column selects into a host variable.

Updatable Views Support

Adaptive Server Enterprise and SQL Anywhere are more liberal than ANSI permits on the view definitions that are updatable when the WITH CHECK option is not requested.

SQL Anywhere offers the **ANSI_UPDATE_CONSTRAINTS** option to control whether updates are restricted to those supported by SQL92, or a more liberal set of rules.

SAP Sybase IQ permits **UPDATE** only on single-table views that can be flattened. SAP Sybase IQ does not support **WITH CHECK**.

Support for FROM Clause in UPDATE and DELETE

SAP Sybase IQAdaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ all support the FROM clause with multiple tables in UPDATE and DELETE.

Transact-SQL Procedure Language Overview

The stored procedure language is the part of SQL used in stored procedures and batches.

SQL Anywhere and SAP Sybase IQ support a large part of the Transact-SQL stored procedure language in addition to the Watcom-SQL dialect based on SQL92.

Transact-SQL Stored Procedure Overview

Because it is based on the ISO/ANSI draft standard, the SQL Anywhere and SAP Sybase IQ stored procedure language differs from the Transact-SQL dialect in many ways.

Many of the concepts and features are similar, but the syntax is different. SQL Anywhere and SAP Sybase IQ support for Transact-SQL takes advantage of the similar concepts by providing automatic translation between dialects. However, you must write a procedure exclusively in one of the two dialects, not in a mixture of the two.

There are a variety of aspects to SQL Anywhere and SAP Sybase IQ support for Transact-SQL stored procedures, including:

- Passing parameters
- Returning result sets
- Returning status information
- Providing default values for parameters
- Control statements
- Error handling

Transact-SQL Batch Overview

In Transact-SQL, a batch is a set of SQL statements submitted together and executed as a group, one after the other.

Batches can be stored in command files. The ISQL utility in SQL Anywhere and SAP Sybase IQ and the `isql` utility in Adaptive Server Enterprise provide similar capabilities for executing batches interactively.

The control statements used in procedures can also be used in batches. SQL Anywhere and SAP Sybase IQ support the use of control statements in batches and the Transact-SQL-like use of nondelimited groups of statements terminated with a **GO** statement to signify the end of a batch.

For batches stored in command files, SQL Anywhere and SAP Sybase IQ support the use of parameters in command files. Adaptive Server Enterprise does not support parameters.

SQL Statements in Procedures and Batches

Some SQL statements supported by SAP Sybase IQ are part of one dialect, but not the other.

You cannot mix the two dialects within a procedure or batch. This means that:

- You can include Transact-SQL-only statements with statements that are part of both dialects in a batch or procedure.
- You can include statements not supported by Adaptive Server Enterprise with statements that are supported by both servers in a batch or procedure.
- You cannot include Transact-SQL–only statements with SAP Sybase IQ—only statements in a batch or procedure.

SQL statements not separated by semicolons are part of a Transact-SQL procedure or batch. See *Reference: Statements and Options* for details of individual statements.

Transact-SQL compatibility has improved; incorrect SQL syntax that was previously accepted now fails with an error.

Expression Subqueries in IF Statements

Adaptive Server Enterprise and SQL Anywhere support comparisons between a variable and a scalar value returned by an expression subquery.

For example:

```
create procedure testIf ()
  begin
  declare var4 int;
set var4 = 10;
  if var4 = (select MIN (a_i1) from a) then set
      var4 = 100;
end if;
end;
```

CASE Statement Support

Permitted usage of the CASE statement differs in SAP Sybase IQ and SQL Anywhere.

The **CASE** statement is not supported in Adaptive Server Enterprise, which supports case expressions only.

See also

- *Expressions* on page 27

Row-level Cursor Operations Support

Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ support the use of cursors with UPDATE and DELETE.

Consider this example:

```
UPDATE WHERE CURRENT OF {cursor}
```

```
DELETE WHERE CURRENT OF {cursor}
```

In SAP Sybase IQ, updatable cursors are asensitive only, for one table only, and chained only. Updatable hold cursors are not permitted. Updatable cursors in SAP Sybase IQ get a table lock.

PRINT Command Support

Support for PRINT differs in Adaptive Server Enterprise, SQL Anywhere, and SAP Sybase IQ.

The effect of **PRINT** depends on the client:

- Adaptive Server Enterprise **PRINT** always sends a message to the client.
- In SQL Anywhere and SAP Sybase IQ, **PRINT** sends a message to the client for Open Client and JDBC connections.
- Adaptive Server Enterprise stored procedures that rely on **PRINT** work in SAP Sybase IQ using Interactive SQL.

Note: SAP Sybase IQ users might prefer Interactive SQL with JDBC, rather than the iAdaptive Server Anywhere JDBC driver (formerly called the JDBC-ODBC bridge).

Automatic Translation of Stored Procedures

In addition to supporting Transact-SQL alternative syntax, SQL Anywhere and SAP Sybase IQ provide aids for translating statements between the Watcom-SQL and Transact-SQL dialects.

Functions returning information about SQL statements and enabling automatic translation of SQL statements include:

Table 156. Functions enabling automatic translation

Function	Description
SQLDialect(statement)	Returns Watcom-SQL or Transact-SQL.
WatcomSQL(statement)	Returns the Watcom-SQL syntax for the statement.
TransactSQL(statement)	Returns the Transact-SQL syntax for the statement.

These are functions and thus can be accessed using a **SELECT** statement from ISQL. For example, the following statement returns the value Watcom-SQL:

```
SELECT SqlDialect('select * from Employees')
```

Result Sets from Transact-SQL Procedures

SQL Anywhere, SAP Sybase IQ procedures and Transact-SQL procedures return result sets differently.

SQL Anywhere and SAP Sybase IQ use a **RESULT** clause to specify returned result sets.

In Transact-SQL procedures, column names or alias names of the first query are returned to the calling environment.

The following Transact-SQL procedure illustrates how Transact-SQL stored procedures return result sets:

```
CREATE PROCEDURE showdept (@deptname varchar(30))
AS
    SELECT Employees.Surname, Employees.GivenName
    FROM Departments, Employees
    WHERE Departments.DepartmentName = @deptname
    AND Departments.DepartmentID = Employees.DepartmentID
```

The following is the corresponding SQL Anywhere or SAP Sybase IQ procedure:

```
CREATE PROCEDURE showdept(in deptname varchar(30))
RESULT ( lastname char(20), firstname char(20))
BEGIN
    SELECT Employees.Surname, Employees.GivenName
    FROM Departments, Employees
    WHERE Departments.DepartmentName = deptname
    AND Departments.DepartmentID = Employee.DepartmentID
END
```

There are minor differences in the way the client tools present multiple results to the client:

- **isql** displays all results in a single stream.

- Interactive SQL presents each result set on a separate tab. You must enable this functionality in the Option menu. Make it a permanent change, then restart or reconnect to Interactive SQL.

Variables in Transact-SQL Procedures

SQL Anywhere and SAP Sybase IQ assign values to variables in procedures differently than Transact-SQL.

SQL Anywhere and SAP Sybase IQ use the SET statement to assign values to variables in a procedure.

In Transact-SQL, values are assigned using the **SELECT** statement with an empty table list. The following simple procedure illustrates how the Transact-SQL syntax works:

```
CREATE PROCEDURE multiply
                    @mult1 int,
                    @mult2 int,
                    @result int output
AS
SELECT @result = @mult1 * @mult2
```

This procedure can be called as follows:

```
CREATE VARIABLE @product int
go
EXECUTE multiply 5, 6, @product OUTPUT
go
```

The variable *@product* has a value of 30 after the procedure executes.

There are some differences in order and persistence of variable declarations:

- In Adaptive Server Enterprise, you can declare variables anywhere in the body of a stored procedure. Variables persist for the duration of the procedure.
- In SQL Anywhere and SAP Sybase IQ, you must declare variables at the beginning of a compound statement (that is, immediately after **BEGIN** in a **BEGIN...END** pair). Variables persist only for the duration of the compound statement.

See also

- *Criteria for Writing Compatible Queries* on page 832

Error Handling in Transact-SQL Procedures

Default procedure error handling is different in the Watcom-SQL and Transact-SQL dialects.

By default, Watcom-SQL dialect procedures exit when they encounter an error, returning SQLSTATE and SQLCODE values to the calling environment.

You can build explicit error handling into Watcom-SQL stored procedures using the **EXCEPTION** statement, or you can instruct the procedure to continue execution at the next statement when it encounters an error, using the **ON EXCEPTION RESUME** statement.

When a Transact-SQL dialect procedure encounters an error, execution continues at the following statement. The global variable **@@error** holds the error status of the most recently executed statement. You can check this variable following a statement to force return from a procedure. For example, the following statement causes an exit if an error occurs:

```
IF @@error != 0 RETURN
```

When the procedure completes execution, a return value indicates the success or failure of the procedure. This return status is an integer, and can be accessed as follows:

```
DECLARE @status INT
EXECUTE @status = proc_sample
IF @status = 0
    PRINT 'procedure succeeded'
ELSE
    PRINT 'procedure failed'
```

This table describes the built-in procedure return values and their meanings:

Table 157. Built-in procedure return values

Value	Meaning
0	Procedure executed without error
-1	Missing object
-2	Data type error
-3	Process was chosen as deadlock victim
-4	Permission error
-5	Syntax error
-6	Miscellaneous user error
-7	Resource error, such as out of space
-8	Nonfatal internal problem
-9	System limit was reached
-10	Fatal internal inconsistency
-11	Fatal internal inconsistency
-12	Table or index is corrupt
-13	Database is corrupt
-14	Hardware error

The **RETURN** statement can be used to return other integers, with their own user-defined meanings.

Using the RAISERROR Statement in Procedures

The RAISERROR statement is a Transact-SQL statement for generating user-defined errors. It has a similar function to the **SIGNAL** statement.

By itself, **RAISERROR** does not cause an exit from the procedure, but it can be combined with a **RETURN** statement or a test of the **@@error** global variable to control execution following a user-defined error.

If you set the **ON_TSQL_ERROR** database option to **CONTINUE**, **RAISERROR** no longer signals an execution-ending error. Instead, the procedure completes and stores the **RAISERROR** status code and message, and returns the most recent **RAISERROR**. If the procedure causing the **RAISERROR** was called from another procedure, **RAISERROR** returns after the outermost calling procedure terminates.

You lose intermediate **RAISERROR** statuses and codes when the procedure terminates. If, at return time, an error occurs along with **RAISERROR**, the error information is returned and you lose the **RAISERROR** information. The application can query intermediate **RAISERROR** statuses by examining **@@error** global variable at different execution points.

Transact-SQL-like Error Handling in the Watcom-SQL Dialect

You can make a Watcom-SQL dialect procedure handle errors in a Transact-SQL-like manner.

Supply the **ON EXCEPTION RESUME** clause to the **CREATE PROCEDURE** statement:

```
CREATE PROCEDURE sample_proc()
ON EXCEPTION RESUME
BEGIN
    ...
END
```

The presence of an **ON EXCEPTION RESUME** clause prevents explicit exception handling code from being executed, so avoid using these two clauses together.

SQL Anywhere and SAP Sybase IQ Differences and Shared Functionality

SAP Sybase IQ and SQL Anywhere have differences in starting and managing databases and servers, database option support, DDL support, and DML support.

For additional information, always refer to the SAP Sybase IQ documentation set when using the product. Refer to the SQL Anywhere documentation set when using SQL Anywhere, or when the SAP Sybase IQ documentation refers to SQL Anywhere documentation for specific functionality only.

SQL Anywhere Server and Database Startup and Administration

Starting and managing databases and servers differs between SAP Sybase IQ and SQL Anywhere.

- SAP Sybase IQ uses the server startup command **start_iq**, instead of the SQL Anywhere network server startup command.
- SAP Sybase IQ does not support personal servers.
- SAP Sybase IQ supports many SQL Anywhere server command line options, but not all. Other server options are supported for SAP Sybase IQ but not for SQL Anywhere.
- SAP Sybase IQ provides the **stop_iq** utility (UNIX) to shut down servers.
- Clauses permitted in the **BACKUP** and **RESTORE** statements differ in SAP Sybase IQ and SQL Anywhere.
- SQL Remote is supported in SAP Sybase IQ only for multiplex operations.

SAP Sybase IQ supports many SQL Anywhere database administration utilities, but not all:

- The following SQL Anywhere utilities are not supported by SAP Sybase IQ:
  - **backup**
  - **compression**
  - **console**
  - **initialization**
  - **license**
  - **log transfer**
  - **log translation**
  - **rebuild**
  - **spawn**
  - some **transaction log** options (**-g**, **-il**, **-ir**, **-n**, **-x**, **-z**)
  - **uncompression**
  - **unload**
  - **upgrade**
  - **write file**
- SAP Sybase IQ supports the SQL Anywhere **validation** utility only on the catalog store. To validate the IQ main store, use **sp_iqcheckdb**.

SQL Anywhere Data Definition Language (DDL) Differences

SQL Anywhere and SAP Sybase IQ have differences in DDL behavior.

- In a **DELETE/DROP** or **PRIMARY KEY** clause of an **ALTER TABLE** statement, SAP Sybase IQ takes the **RESTRICT** action (reports an error if there are associated foreign keys). SQL Anywhere always takes the **CASCADE** action.

- Similarly, **DROP TABLE** statement reports an error in SAP Sybase IQ if there are associated foreign-key constraints.
- SAP Sybase IQ does not support these DDL statements: **CREATE COMPRESSED DATABASE**, **CREATE TRIGGER**, **SETUSER**.
- SAP Sybase IQ supports referential integrity at the statement level, rather than the transaction-level integrity that SQL Anywhere supports with the **CHECK ON COMMIT** clause of the **CREATE TABLE** statement.
- A SAP Sybase IQ table cannot have a foreign key that references a SQL Anywhere (or catalog) table, and a SQL Anywhere table cannot have a foreign key that references a SAP Sybase IQ table.
- In a SAP Sybase IQ database, publications can only be created on SQL Anywhere tables.
- In **CREATE DATABASE**, the defaults for case-sensitivity and collation differ. The defaults for SAP Sybase IQ are CASE RESPECT and the ISO_BINENG collation; for SQL Anywhere, the defaults are CASE IGNORE, and collation inferred from the language and character set of the operating system.
- SAP Sybase IQ does not support the **CREATE ENCRYPTED DATABASE** and **CREATE DECRYPTED DATABASE** commands supported by SQL Anywhere. See *Administration: User Management and Security*.

SQL Anywhere Data Manipulation Language (DML) Differences

Not all SQL Anywhere DML objects and syntax are supported by SAP Sybase IQ.

- SAP Sybase IQ does not support these DML and procedural statements:
  - **EXPLAIN**
  - **GET DATA**
  - **INPUT**
  - **PREPARE TO COMMIT**
  - **PUT**
  - **READTEXT**
  - **ROLLBACK TRIGGER**
  - **SYSTEM**
  - **UNLOAD TABLE**
  - **VALIDATE TABLE**

  Note: A set of extraction options perform a role similar to **UNLOAD TABLE**.
- SAP Sybase IQ supports the **INSERT...LOCATION** syntax; SQL Anywhere does not.
- **LOAD TABLE** options differ in SAP Sybase IQ and SQL Anywhere.
- **OPEN** statement in SAP Sybase IQ does not support **BLOCK** and **ISOLATION LEVEL** clauses.
- SAP Sybase IQ does not support triggers.
- Use of transactions, isolation levels, checkpoints, and automatically generated COMMITs, as well as cursor support, is different in SAP Sybase IQ and SQL Anywhere.

- When you **SELECT** from a stored procedure in SAP Sybase IQ, CIS functional compensation performance considerations apply.
- SAP Sybase IQ ignores the database name qualifier in fully qualified names in Adaptive Server Enterprise **SELECT** statements, such as a **FROM** clause with *`<database name>.<owner>.<table name>`*. For example, SAP Sybase IQ interprets the query `SELECT * FROM XXX..TEST` as `SELECT * FROM TEST`.

Adaptive Server Enterprise and SAP Sybase IQ Differences and Shared Functionality

SAP Sybase IQ and Adaptive Server Enterprise have differences in stored procedure support and views support.

For additional information, always refer to the SAP Sybase IQ documentation set when using the product. Refer to the Adaptive Server Enterprise documentation set when using Adaptive Server Enterprise, or when the SAP Sybase IQ documentation refers to Adaptive Server Enterprise documentation for specific functionality only.

Adaptive Server Enterprise Stored Procedures

Certain stored procedures are not supported by SAP Sybase IQ.

SAP Sybase IQ no longer supports these Adaptive Server Enterprise stored procedures:

- **sp_addserver**
- **sp_configure**
- **sp_estspace**
- **sp_help**
- **sp_helpuser**
- **sp_who**

SAP Sybase IQ no longer supports the following catalog procedures:

- **sp_column_privileges**
- **sp_databases**
- **sp_datatype_info**
- **sp_server_info**

Adaptive Server Enterprise System Views

Certain views are not supported by SAP Sybase IQ.

SAP Sybase IQ no longer supports these Adaptive Server Enterprise views:

- `sysalternates`
- `sysaudits`

- `sysauditoptions`
- `sysconstraints`
- `syscharsets`
- `sysconfigures`
- `syscurconfigs`
- `sysdatabases`
- `sysdepends`
- `sysdevices`
- `sysengines`
- `syskeys`
- `syslanguages`
- `syslocks`
- `syslogs`
- `sysloginroles`
- `sysmessages`
- `sysprocedures`
- `sysprocesses`
- `sysprotects`
- `sysreferences`
- `sysremotelogins`
- `sysroles`
- `syssegments`
- `sysservers`
- `syssrvroles`
- `systhresholds`
- `sysusages`

Column Name Differences

The column name used in the Adaptive Server Enterprise view SYSTYPES is “allownulls”. The column name used in the SAP Sybase IQ view SYSTYPES is “allowsnulls”.

Index