Administration: Load Management

SAP Sybase IQ 16.0 SP2

DOCUMENT ID: DC01773-01-1602-01
LAST REVISED: November 2013

Contents

Read Me First

Although the SAP® Sybase® IQ 16 New Features Summary describes all new SAP Sybase IQ functionality, some features may require additional action on your part to take advantage of the new architecture.

Customers upgrading from a previous release, for example, may need to change some initial compatibility options or rebuild wide columns to accommodate different datatypes. The new load engine provides better performance, but requires changes to the default memory allocation to use all available hardware resources efficiently.

NBit

Continuous `NBit` dictionary compression replaces 1, 2, 3 byte dictionary compression as the default column storage mechanism in 16.0. All datatypes except LOB (character and binary) and BIT datatypes can be `NBit` columns.

The `IQ UNIQUE` column constraint determines whether a column loads as `Flat FP` or `NBit FP`. An `IQ UNIQUE` *n* value set to 0 loads the column as `Flat FP`. An *n* value greater than 0 but less than the `FP_NBIT_AUTOSIZE_LIMIT` creates a `NBit` column initially sized to *n*. Columns without an `IQ UNIQUE` constraint implicitly load as `NBit` up to the auto-size limit.

Using `IQ UNIQUE` with an *n* value less than the auto-size limit is not necessary. The load engine automatically sizes all low or medium cardinality columns as `NBit`. Use `IQ UNIQUE` in cases where you want to load the column as `Flat FP` or when you want to load a column as `NBit` when the number of distinct values exceeds the auto-size limits.

Loads and Large Memory

Large memory represents the maximum amount of memory that SAP Sybase IQ can dynamically request from the OS for temporary use. Because some load operations may require more large memory than the 2GB default provides, adjust the startup options that control large and cache memory allocation based on the total amount of available physical memory.

As a general rule, large memory requirements represent one third of the total available physical memory allocated to SAP Sybase IQ. To ensure adequate memory for the main and temporary IQ stores, set the **–iqlm**, **–iqtc**, and **–iqmc** startup parameters so that each parameter receives one third of all available physical memory allocated to SAP Sybase IQ.

In most cases, you should allocate 80% of total physical memory to SAP Sybase IQ to prevent SAP Sybase IQ processes from being swapped out. Adjust actual memory allocation to accommodate other processes running on the same system. For example, on a machine with 32 cores and 128GB of total available physical memory, you would allocate 100GB

(approximately 80% of the 128GB total) to SAP Sybase IQ processes. Following the general rule, you would set the **–iqlm**, **–iqtc**, and **–iqmc** parameters to 33GB each.

See *-iqlm iqsrv16 Server Option* and *-iqmc iqsrv16 Server Option* in the *Utility Guide*.

Index Changes

Changes to `FP` and `HG` indexes take advantage of the new column compression mechanism and improve load performance.

Index	Description
New Fast Projection (FP) Indexes	Take advantage of the new continuous `NBit` dictionary compression, which replaces `FP(1)`, `FP(2)`, and `FP(3)` byte dictionary compression. `FP(1)`, `FP(2)`, and `FP(3)` indexes roll over to `NBit(8)`, `NBit(16)`, and `NBit(24)` respectively. If `FP_NBIT_IQ15_COMPATIBILITY='OFF'`, `IQ UNIQUE` constraints applied to the column determine whether the column loads as `Flat FP` or `NBit`. See *Fast Projection (FP) Index* in *Administration: Database*.
New tiered HG index structure	Decouples load performance from `HG` index size. In 15.x, load throughput could degrade as the amount of data in an `HG` index increased. As the index grew, loading the same amount of data could take more time. The new tiered structure decouples load performance from the `HG` index size to increase throughput. The `CREATE_HG_WITH_EXACT_DISTINCTS` option determines whether newly created `HG` indexes are tiered or non-tiered. This option is ON in all new 16.0 databases and all 16.0 databases migrated from 15.x. To take advantage of the new structure, set this option to OFF. Use **sp_iqrebuildindex** to convert non-tired `HG` indexes to tiered `HG` and vice-versa. See *CREATE_HG_WITH_EXACT_DISTINCTS Option* in *Reference: Statements and Options* .

Stored Procedures

New stored procedures return information about column indexes and constraints.

Procedure	Description
sp_iqindexmetadata	Returns details about column indexes, including the index types (`Flat FP`, `NBit`, `HG`, and tiered `HG`), distinct counts, `IQ UNIQUE` *n* value, and `NBit` dictionary size. See *sp_iqindexmetadata Procedure* in *Reference: Building Blocks, Tables, and Procedures*
sp_iqcolumnmetadata	Returns `FP` index metadata for one or more user tables or all tables in the database. See *sp_iqcolumnmetadata Procedure* in *Reference: Building Blocks, Tables, and Procedures*
sp_iqindexrebuildwidedata	Identifies wide columns that you must rebuild before they are available for read/write activities. Output includes statements that you can use with **sp_iqrebuildindex** to rebuild the columns. See *sp_iqindexrebuildwidedata Procedure* in *Reference: Building Blocks, Tables, and Procedures*
sp_iqrebuildindex	Rebuilds `FP` indexes (`Flat FP as NBit`, or `NBit` as `Flat FP`) and `HG` indexes (single `HG` as tiered `HG`, or tiered `HG` as single `HG`). Before you can insert or update new data, you must rebuild all columns greater than 255 bytes wide. The `index_clause` can reset `IQ UNIQUE` *n* to an explicit value from 0 (to recast an `NBit` column to `Flat FP`) up to the limits defined in the **FP_NBIT_AUTOSIZE_LIMIT** and **FP_NBIT_LOOKUP_MB** options. **sp_iqrebuildindex** also enables read-write access to columns that contain large object (LOB) data. LOB columns migrated from 15.x databases are read-only until you run **sp_iqrebuildindex**.The estimated cardinality for `NBit` columns with an `IQ UNIQUE` value below or equal to the `FP_NBIT_AUTOSIZE_LIMIT` is stored as 0 regardless of the `FP_NBIT_IQ15_COMPATIBILITY` setting. This affects the value returned from **sp_iqindexmetadata**. See *sp_iqrebuildindex Procedure* in *Reference: Building Blocks, Tables, and Procedures*

Procedure	Description
sp_iqrebuildindexwide	SAP Sybase IQ implicitly rebuilds 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 the first time a pre-16.0 non-RLV base table is opened for read-write access. You can also explicitly rebuild wide columns using the **sp_iqrebuildindexwide** procedure. **sp_iqrebuildindexwide** can rebuild wide tables by **table_name**, **table_owner**, and **level**. Depending on the argument, this procedure can rebuild all pre-16.0 columns wider than 255 bytes, some or all tokenized FPs, VARCHAR/VARBINARY columns, and all fixed Flat FPs for specified tables in the database. See *sp_iqrebuildindexwide Procedure* in *Reference: Building Blocks, Tables, and Procedures*.

Database Options

Some database options are not enabled to take advantage of 16.0 features. Maintaining limited compatibility after a database upgrade provides some flexibility to transition existing applications.

Option	Description
FP_NBIT_IQ15_COMPATIBILITY	Provides tokenized `FP` support similar to that available in 15.x. This option is ON by default in all 16.0 databases upgraded from 15.x and OFF in all newly created 16.0 databases. • If this option is ON, the database engine uses the `MINIMIZE_STORAGE`, `FP_LOOKUP_SIZE`, and `FP_LOOKUP_SIZE_PPM` options to optimize column storage. These options are ignored in 16.0. • If this option is OFF, the database engine ignores 15.x options and columns conform to SAP Sybase IQ `NBit` storage options. Set this option to OFF to take advantage of `NBit` column compression.

Option	Description
CREATE_HG_WITH_EXACT_DISTINCTS	Determines whether new HG indexes explicitly created with a **CREATE INDEX** command, or implicitly creating or altering a table with a PRIMARY KEY or a FOREIGN KEY declaration, are tiered or non-tiered. This option is ON 16.0 in all databases upgraded from 15.x and all newly created 16.0 databases. If this option is ON, all new HG indexes are non-tiered. To take advantage of the new tiered HG index structure, set this option to OFF. Use **sp_iqrebuildindex** to convert non-tiered HG indexes to tiered HG and vice-versa.
CREATE_HG_AND_FORCE_PHYSICAL_DELETE	Governs 16.0 delete behavior for tiered HG indexes. This option determines whether SAP Sybase IQ performs a physical delete immediately or defers the delete to a point later in the load. **CREATE_HG_AND_FORCE_PHYSICAL_DELETE** is ON by default, which instructs SAP Sybase IQ to perform physical deletes.
REVERT_TO_V15_OPTIMIZER	**REVERT_TO_V15_OPTIMIZER** forces the query optimizer to mimic SAP Sybase IQ 15.x behavior. **REVERT_TO_V15_OPTIMIZER**='ON' by default in all 16.0 databases upgraded from 15.x. **REVERT_TO_V15_OPTIMIZER**='OFF' by default in all newly created SAP Sybase IQ 16.0 databases. If you plan to use SAP Sybase IQ hash partitioning features, set the **REVERT_TO_V15_OPTIMIZER**='OFF' in databases upgraded from 15.x to 16.0.

Import and Export Overview

SAP Sybase IQ lets you import data from flat files or directly from database tables. You can also enter specified values directly into the database. You can export data to other formats from the Interactive SQL utility and the IQ data extraction facility.

SAP Sybase IQ tables are logical table; they do not contain data. All the information needed to resolve queries, including data, is contained in the SAP Sybase IQ indexes. When you insert data into the columns in an IQ table, you are not actually adding data to columns in the table, but rather to the column indexes. You build indexes by inserting data on a table-by-table basis.

Import and Export Method Selection

SAP Sybase IQ offers you a choice of methods for adding, changing, or deleting data.

- For efficient bulk loading of tables from flat files, use the SQL statement **LOAD TABLE**.
- To insert specified values into a table row by row, use the SQL statement **INSERT** with the **VALUES** option.
- To insert rows selected from a table (including a table residing in another database), use the SQL statement **INSERT** with a **SELECT** statement clause.
- To remove specific rows from a table, use the **DELETE** statement.
- To change existing rows in a table, use the **UPDATE** statement.

The IQ data extraction facility exports data in binary or ASCII format, which you can then load into another database. Use this facility for high-volume data movement, or when you need an output file that can be used for loads.

From Interactive SQL, you can export data to another database in a variety of formats, or produce a text file as output. You can also redirect the output of any command.

Note: SAP Sybase IQ supports BCP through the **LOAD TABLE FORMAT BCP** option. You can directly perform a BCP into an IQ table. SAP Sybase IQ also supports bulk loading of remote data using the **LOAD TABLE USING CLIENT FILE** option.

See also

- *Input and Output Data Formats* on page 8
- *Permissions for Modifying Data* on page 8
- *Schedule Database Updates* on page 9

Input and Output Data Formats

The **LOAD TABLE** statement imports data from files row by row. Both ASCII and binary input files are supported, with either fixed-length fields or variable-length fields ended by a delimiter.

The **INSERT** statement moves data into a SAP Sybase IQ table either from a specified set of values, or directly from tables.

OUTPUT statement file formats supported by Interactive SQL include:

- **TEXT**
- **FIXED**
- **HTML**
- **SQL**
- **XML**

The IQ data extraction facility exports data in binary or ASCII format.

See also

- *Import and Export Method Selection* on page 7
- *Permissions for Modifying Data* on page 8
- *Schedule Database Updates* on page 9

Specifying an Output Format for Interactive SQL

You can specify a default output format from Interactive SQL.

1. In the SQL Statements window, select **Options**.
2. Select **Import/Export**.
3. From the drop down list, select a default export format.
4. From the drop down list, select a default import format.

Permissions for Modifying Data

You can execute data modification statements only if you have the proper permissions on the database tables you want to modify.

The database administrator and the owners of database objects use the GRANT and REVOKE statements to decide who has access to which data modification functions.

To insert data, you need INSERT permission for that table or view. To delete data, you need DELETE permission for that table or view. To update data, you need UPDATE permission.

The DBA can insert into or delete from any table. The owner of a table has INSERT, DELETE, and UPDATE permission on it.

Permissions can be granted to and revoked from individual users, roles, or the PUBLIC role.

See also

- *Import and Export Method Selection* on page 7
- *Input and Output Data Formats* on page 8
- *Schedule Database Updates* on page 9

Schedule Database Updates

Multiple users can query a database table, and can update the database concurrently.

See also

- *Import and Export Method Selection* on page 7
- *Input and Output Data Formats* on page 8
- *Permissions for Modifying Data* on page 8

Methods for Exporting Data from a Database

There are several ways to export data from your database, including output redirection and using a data extraction facility.

You may also export data by using a front-end tool, written by you or a third party, that effectively queries the IQ database and formats the data as desired.

Output Redirection

You can use output redirection to export query results.

You can redirect the output of any command to a file or device by putting the **>#** redirection symbol anywhere on the command. The redirection symbol must be followed by a file name. (In a command file, the file name is then followed by the semicolon used as statement terminator.) The file is placed relative to the directory where Interactive SQL was started.

This example redirects output to a file called `empfile`:

```
SELECT *
FROM Employees
>#  empfile
```

Do not enclose the file name in quotation marks.

Output redirection is most useful on the **SELECT** statement.

Use two **>** characters in a redirection symbol instead of one (for example, **>>#**), to append output to the specified file instead of replacing the contents of the file. Headings are included in the output from the **SELECT** statement if the output starts at the beginning of the specified file and the output format supports headings.

Redirecting Output and Messages

The **>&** redirection symbol redirects all output including error messages and statistics for the command on which it appears. For example:

```
SELECT *
FROM Employees
>&  empfile
```

Do not enclose the file name in quotation marks.

This example sends the **SELECT** statement to the file `empfile`, followed by the output from the statement, and some statistics pertaining to the command.

You can use the *>&* redirection method to obtain a log of what happens during a **READ** command. The statistics and errors of each command are written following the command in the redirected output file.

NULL Value Output

Although the most common reason to extract data is for use in other software products, these products may sometimes have issues processing NULL values.

The **dbisql** option **NULLS** allows you to choose how NULL values are output. Alternatively, you can use the **IFNULL** function to output a specific value whenever there is a NULL value.

Data Extraction Facility

The data extraction facility is a group of database options that dramatically improve performance for queries with large result sets.

Like other database options, you can set the data extraction options as temporary or permanent. Ordinarily, these options are set as temporary, for a specific connection.

Advantages of using the extraction options include:

- A binary format is supported, which allows loading the output data into the same or a different IQ database.
- A **SELECT** statement with heavy output runs up to four times faster for ASCII output, and up to nine times faster for binary output.

Extract Options

The extract options let you redirect the output of a **SELECT** statement from the standard interface to go directly to one or more disk files or named pipes.

Option Name	Allowed Values	Default Value
Temp_Extract_Append	ON or OFF	OFF
Temp_Extract_Binary	ON or OFF	OFF
Temp_Extract_Column_Delimiter	String	','
Temp_Extract_Directory	String	''
Temp_Extract_Name1	String	''
Temp_Extract_Name2	String	''
Temp_Extract_Name3	String	''
Temp_Extract_Name4	String	''

Option Name	Allowed Values	Default Value
Temp_Extract_Name5	String	"
Temp_Extract_Name6	String	"
Temp_Extract_Name7	String	"
Temp_Extract_Name8	String	"
Temp_Extract_Null_As_Empty	ON or OFF	OFF
Temp_Extract_Null_As_Zero	ON or OFF	OFF
Temp_Extract_Quote	String	"
Temp_Extract_Quotes	ON or OFF	OFF
Temp_Extract_Quotes_All	ON or OFF	OFF
Temp_Extract_Row_Delimiter	String	"
Temp_Extract_Size1	Platform specific*	0
Temp_Extract_Size2	Platform specific*	0
Temp_Extract_Size3	Platform specific*	0
Temp_Extract_Size4	Platform specific*	0
Temp_Extract_Size5	Platform specific*	0
Temp_Extract_Size6	Platform specific*	0
Temp_Extract_Size7	Platform specific*	0
Temp_Extract_Size8	Platform specific*	0
Temp_Extract_Swap	ON or OFF	OFF

*The default values for the `TEMP_EXTRACT_SIZEn` options are platform specific:

- AIX and HP-UX: 0 – 64GB
- Sun Solaris: 0 – 512GB
- Windows: 0 – 128GB
- Linux: 0 – 512GB

When large file systems, such as JFS2, support file sizes larger than the default value, set `TEMP_EXTRACT_SIZEn` to the maximum value that the file system allows. For example, to support 1TB set option, enter:

```
SET OPTION TEMP_EXTRACT_SIZE1 = 1073741824 KB
```

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

The most important of these options is TEMP_EXTRACT_NAME1; if it is set to its default setting (the empty string), extraction is disabled and no output is redirected. To enable extraction, set TEMP_EXTRACT_NAME1 to a path name. Choose a path and file name that are not otherwise in use. If the file does not already exist, the data extraction facility creates the file.

Both the directory / folder containing the named file, and the named file itself, must have write permission set for the user who started IQ (for example, *sybase*). In append mode, the data extraction facility adds extracted rows to the end of the file and does not overwrite the data that is already in the file.

Warning! If you choose the path name of an existing file and the TEMP_EXTRACT_APPEND option is OFF (the default), file contents are overwritten.

Use the options TEMP_EXTRACT_NAME2 through TEMP_EXTRACT_NAME8 to specify the names of multiple output files. You must use options sequentially. For example, TEMP_EXTRACT_NAME3 has no effect unless both TEMP_EXTRACT_NAME1 and TEMP_EXTRACT_NAME2 are already set.

Use TEMP_EXTRACT_SIZE1 through TEMP_EXTRACT_SIZE8 to specify the maximum size of the corresponding output files. TEMP_EXTRACT_SIZE1 specifies the maximum size of the output file specified by TEMP_EXTRACT_NAME1, TEMP_EXTRACT_SIZE2 specifies the maximum size of the output file specified by TEMP_EXTRACT_NAME2, and so on.

The default minimum for the data extraction size options is 0. IQ converts this default to the following values:

Device Type	Size
Disk File	AIX and HP-UX: 0 – 64GB Sun Solaris & Linux: 0 – 512GB Windows: 0 – 128GB
Other	Unlimited

TEMP_EXTRACT_APPEND is incompatible with the TEMP_EXTRACT_SIZEn options. If you try to restrict the size of the extract append output file, SAP Sybase IQ reports an error.

If you are extracting to a single disk file or a single named pipe, leave TEMP_EXTRACT_NAME2 through TEMP_EXTRACT_NAME8, and TEMP_EXTRACT_SIZE1 through TEMP_EXTRACT_SIZE8 at their default values.

Note: If the **SELECT** returns no rows and there is no output to redirect, an empty file of zero length is created. If you specify multiple extract files and there is not enough data to fill all of the files, all of the files are still created.

Controlling Access

The TEMP_EXTRACT_DIRECTORY option controls whether a user is allowed to use the data extraction facility. It also controls the directory into which temporary extraction files are placed and overrides a directory path specified in the TEMP_EXTRACT_NAMEn options.

If the TEMP_EXTRACT_DIRECTORY option is set to the string FORBIDDEN (case-insensitive) for a user, then that user is not allowed to perform data extracts. Any attempt to do so results in the error: `You do not have permission to perform Extracts`.

If TEMP_EXTRACT_DIRECTORY is set to FORBIDDEN for the PUBLIC role, no one can run data extraction.

If TEMP_EXTRACT_DIRECTORY is set to a valid directory path, temporary extraction files are placed in the specified directory, overriding paths in the TEMP_EXTRACT_NAMEn options.

If TEMP_EXTRACT_DIRECTORY is set to an invalid directory path, an error occurs: `File does not exist File: <invalid path>.`

If TEMP_EXTRACT_DIRECTORY is blank, then temp extract files are placed in directories according to their specification in TEMP_EXTRACT_NAMEn. If no path is specified as part of TEMP_EXTRACT_NAMEn, the extract files are, by default, placed in the server startup directory.

The TEMP_EXTRACT_DIRECTORY option provides increased security and helps control disk management by restricting the creation of large data extraction files to the directories for which a user has write access. Setting the option requires the SET ANY SYSTEM OPTION system privilege. This option takes effect immediately.

Types of Extraction

Types of data extraction include:

- Binary
- Binary/swap
- ASCII

A binary extraction produces a file with an overall "binary" format and a per-column "binary with null byte" format. You can use a **LOAD TABLE** statement to load the file.

A binary/swap extraction is the same as a binary extraction, except it is designed to be loaded on another machine with opposite endianness.

An ASCII extraction produces a text file.

The two options Temp_Extract_Binary and Temp_Extract_Swap determine which of the three types of extraction is done:

Type	Temp_Extract_Binary	Temp_Extract_Swap
Binary	ON	OFF
Binary/swap	ON	ON
ASCII	OFF	OFF

The default extraction type is ASCII.

If the data is unloaded using the extraction facility with the `TEMP_EXTRACT_BINARY` option ON, you must use the **LOAD TABLE** statement **BINARY WITH NULL BYTE** parameter for each column when you load the binary data.

Column and Row Delimiters

In an ASCII extraction, the default is to separate column values with commas, and end the row with a newline on UNIX platforms and with a carriage return/newline pair on Windows platforms. The strings are unquoted. If these defaults are unsuitable, change the delimiters, using:

- Temp_Extract_Column_Delimiter
- Temp_Extract_Row_Delimiter
- Temp_Extract_Quote
- Temp_Extract_Quotes
- Temp_Extract_Quotes_All

The delimiter must occupy from 1 - 4 bytes and must be valid in the collation order you are using, if you are using a multibyte collation order. Choose delimiters that do not occur in any of the data output strings themselves.

The default for the Temp_Extract_Row_Delimiter option is '' (an empty string). IQ converts the empty string default for this option to the newline on UNIX platforms and to the carriage return/newline pair on Windows platforms.

The option Temp_Extract_Column_Delimiter controls the delimiter between columns. If this option is set to an empty string for ASCII extractions, the extracted data is written in fixed-width ASCII with no column delimiter. Numeric and binary data types are right-justified on a field of *n* blanks, where *n* is the maximum number of bytes needed for any value of that type. Character data types are left-justified on a field of *n* blanks.

Note: The minimum column width in a fixed-width ASCII extraction is four bytes to allow the string “NULL” for a NULL value. For example, if the extracted column is `CHAR(2)` and Temp_Extract_Column_Delimiter is set to the empty string, there are two spaces after the extracted data.

During ASCII extraction, these control the use of quotes:

Option	ASCII Extraction Action
Temp_Extract_Quotes	String fields enclosed in quotes
Temp_Extract_Quotes_All	All fields enclosed in quotes
Temp_Extract_Quote	Specifies string to be used as the quote

The quote string specified in the Temp_Extract_Quote option has the same restrictions as delimiters. The default for this option is the empty string, which IQ converts to the single quote mark.

Representation of Null Values

TEMP_EXTRACT_NULL_AS_ZERO and TEMP_EXTRACT_NULL_AS_EMPTY control the representation of null values for ASCII extractions. When TEMP_EXTRACT_NULL_AS_ZERO is set to ON, a null value is represented as follows:

- '0' for arithmetic type
- ' ' (the empty string) for the CHAR and VARCHAR character types
- ' ' (the empty string) for dates
- ' ' (the empty string) for times
- ' ' (the empty string) for timestamps

When TEMP_EXTRACT_NULL_AS_EMPTY is set to ON, a null value is represented as ' ' (the empty string) for all data types.

The quotes shown above are not present in the extract output file. When TEMP_EXTRACT_NULL_AS_ZERO and TEMP_EXTRACT_NULL_AS_EMPTY are set to OFF (the default value), the string 'NULL' is used in all cases to represent a NULL value.

If TEMP_EXTRACT_NULL_AS_ZERO is ON, the number of characters that an ASCII extract writes to a file for a `CHAR` or `VARCHAR` column equals the number of characters in the column, even if that number is fewer than four.

Message Logging

When the Query_Plan option is ON, a timestamped list of the extracted columns appears in the IQ message log.

See also

- *Enabling Data Extraction Options* on page 17

Enabling Data Extraction Options

Use the data extraction options with care.

Warning! If you set the extraction options, then execute a **SELECT** statement, and then execute a second **SELECT** statement without changing the extraction file name, the output of

the second **SELECT** overwrites the output of the first **SELECT**. Each time you execute a **SELECT** statement, whether it is one second later or a week later, extraction starts over again, unless the Temp_Extract_Append option is set ON.

The extraction options are set for the connection. If you set the extraction options and another user connects to the database using the same user ID, the extraction facility is also enabled for that user. Your extraction output might be overwritten by another user on the same connection.

Similarly, if another user logs in using the same user ID, the output of queries run by this user is directed to the extraction file until the option is disabled. Run extraction requests using a unique user ID.

1. In a separate location, save any existing output you need to retain.
2. Remove any previously used extraction files.
3. Set the extraction options you require, making sure to set Temp_Extract_Name1 to the file path that is to receive the output.
4. Issue a **SELECT** statement to extraction the data you require.
5. When you finish making extractions, reset Temp_Extract_Name1 to the empty string, or disconnect if set temporarily.

See also

- *Extract Options* on page 12

Data Extraction Option Examples

There are various data extraction scenarios.

Example: Extracting to a Single Disk File

The statements extract to a single disk file `daily_report.txt`:

```
SET TEMPORARY OPTION Temp_Extract_Name1 = 'daily_report.txt';
SET TEMPORARY OPTION Temp_Extract_Name2 = '';
SELECT ....;
SET TEMPORARY OPTION Temp_Extract_Name1 = '';
```

Temp_Extract_Name2 is set to the empty string before the **SELECT** statement is executed, restricting output to a single file.

Temp_Extract_Name1 is set to the empty string after the **SELECT** statement to disable extraction. If extraction is not disabled, then the next **SELECT** statement executed overwrites the `daily_report.txt` file.

Example: Extracting in Append Mode

The disk output file `hourly_report.txt` is already created and has write permission set for the user *sybase*. The following statements extract to `hourly_report.txt`, appending the output from each **SELECT** statement to the end of the file:

```
SET TEMPORARY OPTION Temp_Extract_Append = ON;
SET TEMPORARY OPTION Temp_Extract_Name1 = 'hourly_report.txt';
SET TEMPORARY OPTION Temp_Extract_Name2 = '';
SELECT ....;
SELECT ....;
SELECT ....;
SET TEMPORARY OPTION Temp_Extract_Name1 = '';
```

All output from the three **SELECT** statements is written to `hourly_report.txt`. Temp_Extract_Name1 is set to the empty string after the last **SELECT** statement, to disable extraction. If extraction is not disabled, output from the next **SELECT** statement executed is added to the end of `hourly_report.txt`.

Example: Extracting to Multiple Disk Files

The statements extract to disk files `file1.out`, `file2.out`, and `file3.out`.

First set the file name options:

```
SET TEMPORARY OPTION Temp_Extract_Name1 = 'file1.out';
SET TEMPORARY OPTION Temp_Extract_Name2 = 'file2.out';
SET TEMPORARY OPTION Temp_Extract_Name3 = 'file3.out';
SET TEMPORARY OPTION Temp_Extract_Name4 = '';
```

Now limit the size of the files to 1MB each, by setting the corresponding extract size options:

```
SET TEMPORARY OPTION Temp_Extract_Size1 = '1024';
SET TEMPORARY OPTION Temp_Extract_Size2 = '1024';
SET TEMPORARY OPTION Temp_Extract_Size3 = '1024';
```

The size options are in KB (1024 bytes).

With these settings, the extraction output is first written to `file1.out`. When the next row to be written to `file1.out` would cause the file size to exceed 1MB, the output is redirected to `file2.out`. When `file2.out` is full (writing another row to `file2.out` would cause the file size to exceed 1MB), the output is redirected to `file3.out`. An error is reported, if the size of `file3.out` exceeds 1MB before IQ extracts all rows.

See also

- *Extraction Limitations* on page 19

Extraction Limitations

Restrictions and limitations affect the data extraction facility.

- Extract works only with data stored in the IQ store.
- Extract does not work on system tables or cross database joins.
- Extract does not work with queries that use user-defined functions or system functions, except for the system functions **suser_id()** and **suser_name()**.
- A binary **LOAD TABLE** always trims blanks from `VARCHAR` data. If you have `VARCHAR` data with trailing blanks, they are not preserved on insert by a binary load.

- Trailing zeros are padded onto VARBINARY data during the extract. For example, a field declared as varbinary(6), which contains the data 0x1234, is padded with zeros during extraction and is loaded after extraction as 0x123400.
- To reproduce floating point data exactly, use the binary option.
- Tape devices are not currently supported.
- If you run **dbisql** (Interactive SQL) with the **-q** (quiet mode) option and the data extraction commands are in a command file, you must first set and make permanent the **dbisql** option **Show multiple result sets**. If this option is not set, the output file is not created.

  To set the **Show multiple result sets** option, click **Tools > Options** in the **dbisql** window, then select **Show multiple result sets** and click **Make permanent**.

When Temp_Extract_Name1 is set, you cannot perform **INSERT...SELECT**.

Events do not support execution of statements that return result sets. The server log returns an error similar to:

```
Handler for event 'test_ev' caused SQLSTATE '09W03'
Result set not permitted in 'test_ev'
```

To execute a query through an event, create an event that calls a stored procedure and insert the stored procedure results into a temporary table. If extract is used, the temporary table is always empty and requires little overhead.

For example:

```
CREATE PROCEDURE proc1()
BEGIN
    SET TEMPORARY OPTION temp_extract_name1 = 'testproc.out';
    SELECT * FROM iq_table;
END;

CREATE EVENT "test_ev" ENABLE HANDLER
BEGIN
    SELECT * INTO #tmp FROM proc1();
END;

TRIGGER EVENT test_ev;
```

See also

- *Data Extraction Option Examples* on page 18

Bulk Loads with the LOAD TABLE Statement

The **LOAD TABLE** statement efficiently imports data from a text or binary file into an existing database table, into column indexes created automatically or defined by users.

Set the permissions needed to execute a **LOAD TABLE** statement at the server command line, using the **-gl** option. We recommend the **-gl all** setting, which is the default set by **start_iq**. If **-gl all** is set, you must be the owner of the table, have `ALTER` or `LOAD` permission on the table, or have the `ALTER ANY TABLE`, `LOAD ANY TABLE`, or `ALTER ANY OBJECT` system privilege, to use the **LOAD TABLE** statement. You must also have a write lock on the table.

To load large amounts of data, most users create command files.

Transaction Processing and LOAD TABLE

When you issue the **LOAD TABLE** statement for an IQ table, a savepoint occurs automatically before the data is loaded.

If the load completes successfully, SAP Sybase IQ releases the savepoint. If the load fails, the transaction rolls back to the savepoint. This approach gives you flexibility in committing transactions. For example, if you issue two **LOAD TABLE** commands, you can ensure that either both commands commit or neither commits.

When you issue **LOAD TABLE** for a catalog store table, there is no automatic savepoint. If the load succeeds, it commits automatically. If the load fails, it rolls back. You cannot roll back a successful load of a catalog store table.

Load from a Flat File: UNIX Example

This example assumes that no explicit data conversion is needed, and that the width of input columns matches the width of columns in the `Departments` table. The flat file `dept.txt` must exist at the specified location.

The statement loads the data from the file `dept.txt` into all columns of the `department` table.

```
LOAD TABLE Departments
( DepartmentID, DepartmentName, DepartmentHeadID )
FROM '/d1/MILL1/dept.txt'
```

File Specification Requirements for Loads

In the **FROM** clause, use *filename-string* to specify files, and use commas to separate multiple strings.

The files are read one at a time, and processed in a left-to-right order as specified in the **FROM** clause. Any **SKIP** or **LIMIT** value only applies at the beginning of the load, not for each file.

If a load cannot complete, for example due to insufficient memory, the entire load transaction rolls back.

The *filename-string* is passed to the server as a string, which is subject to the same formatting requirements as other SQL strings. In particular:

- If a backslash (\) precedes the characters `n`, `x`, or `\`, it is considered an escape character. For this reason, to indicate directory paths in Windows systems, you must represent the backslash character by two backslashes if the next character is any of those listed. To load data from the file `c:\newinput.dat` into the `employee` table, use:

  ```
  LOAD TABLE employees
  FROM 'c:\\newinput.dat' ...
  ```
- For server-side loading (**LOAD TABLE... FROM or LOAD TABLE...USING FILE**), the path name is relative to the database server, not to the client application. If you are running the statement on a database server on another computer, the directory name refers to directories on the server machine, not on the client machine. The input file for the load must be on the server machine.
- For client-side data loading (**LOAD TABLE ... USING CLIENT FILE**), the path name must be relative to the client application. The directory name refers to directories on the client machine.

Loads That Specify Named Pipes

When you load from a named pipe on Windows, the program writing to the pipe must close the pipe in a special way. It must call `FlushFileBuffers( )`, then `DisconnectNamedPipe( )`. If the program does not do this, SAP Sybase IQ reports an exception from `hos_io::Read( )`. This issues a `PIPE_NOT_CONNECTED` error, which notifies SAP Sybase IQ that the pipe was shut down in an orderly manner rather than as an uncontrolled disconnect. Refer to the Microsoft documentation for details on these calls.

Loads That Specify Input Data Format

You can specify a wide range of load options that tell SAP Sybase IQ how to interpret and process the input file and what to do when errors occur.

You can specify load options in any order.

Example: Load That Displays Quotation Marks

Consider a table defined as:

```
CREATE TABLE t1 (c1 INT, c2 VARCHAR(20), c3 VARCHAR(20))
```

with the following input data:

```
1, apple , fruit1     ,
2, "banana" , "fruit2",
3, " pear ", " fruit3 ",
```

Execute this query to show the result of loading this data:

```
SELECT c1, c2, c3, LENGTH(c2), LENGTH(c3) FROM t1
```

Given the values of the **QUOTES** and **STRIP** options used during the **LOAD TABLE** command, the following table displays the result of the query, with each result enclosed by angle brackets:

LOAD TABLE Options		**Results of SELECT c1, c2, c3, LENGTH(c2), LENGTH(c3) FROM t1**				
QUOTES	**STRIP**	**c1**	**c2**	**c3**	**length(c2)**	**length(c3)**
ON	RTRIM	<1>	<apple>	<fruit1>	<5>	<6>
		<2>	<banana>	<fruit2>	<6>	<6>
		<3>	< pear >	<fruit3 >	<6>	<8>
ON	OFF	<1>	<apple >	<fruit1 >	<6>	<7>
		<2>	<banana>	<fruit2>	<6>	<6>
		<3>	< pear >	< fruit3 >	<6>	<8>
OFF	RTRIM	<1>	< apple>	< fruit1>	<6>	<7>
		<2>	< “banana”>	< “fruit2”>	<9>	<9>
		<3>	< “ pear ”>	< “ fruit3 ”>	<9>	<11>
OFF	OFF	<1>	< apple >	< fruit1 >	<7>	<8>
		<2>	< “banana” >	< “fruit2”>	<10>	<9>
		<3>	< “ pear ”>	< “ fruit3 ”>	<9>	<11>

Notes on the results:

- With **QUOTES ON** and **STRIP RTRIM**, both leading space and trailing space for `c2` row 1 are trimmed.
- With **QUOTES ON** and **STRIP OFF**, only the leading space for `c2` row 1 is trimmed.
- With **QUOTES OFF** and **STRIP RTRIM**, only the trailing space for `c2` row 1 is trimmed.
- With **QUOTES OFF** and **STRIP OFF**, neither leading space nor trailing space for `c2` row 1 is trimmed.
- With **QUOTES ON** and **STRIP RTRIM**, both leading space and trailing space within quotes for `c2` and `c3` row 3 are NOT trimmed.

Example: Load That Skips Specified Fields.
A Windows example:

```
LOAD TABLE nn
    (l_orderkey,
    l_quantity ASCII(PREFIX 2),
    FILLER(2),
FROM 'C:\\iq\archive\\mill.txt'
BYTE ORDER LOW
```

Example: Load That Limits the Number of Rows Inserted

In this Windows example, no rows are skipped, and 1,000,000 rows are inserted.

```
LOAD TABLE lineitem
    (l_shipmode ASCII(15),
    l_quantity ASCII(8),
    FILLER(30))
FROM 'C:\\iq\archive\\mill.txt'
PREVIEW ON
LIMIT 1000000
```

Example: Load That Includes Tabs and New Lines

The following Windows example sets the column delimiter for the l_orderkey column to tab, and the row delimiter to newline (\x0a) followed by carriage return (\x0d):

```
LOAD TABLE mm
    (l_orderkey '\x09',
    l_quantity ASCII(4),
    FILLER(6),
    l_shipdate DATE('YYYY/MM/DD'))
FROM 'C:\\iq\\archive\\mill.txt'
ROW DELIMITED BY '\x0a\x0d'
```

Example: Load That Skips Rows

In this UNIX example, SAP Sybase IQ reads 9,000 rows from the input file, skips the first 5,000, and loads the next 4,000. If there are only 8,000 rows in the input file, only 3,000 rows are loaded.

```
LOAD TABLE lineitem(
    l_shipmode ASCII(15),
    l_quantity ASCII(8),
FILLER(30))
FROM '/d1/MILL1/tt.t'
LIMIT 4000
SKIP 5000
PREVIEW ON
```

LOAD TABLE Adds Rows

The **LOAD TABLE** statement appends the contents of the file to the existing rows of the table.

To empty an existing table, use the **TRUNCATE TABLE** statement to remove all the rows.

See also

- *Direct Loading of Data from Clients* on page 25
- *Considerations for Partitioned Table Loads* on page 25

- *Load and Insert Messages* on page 26
- *Integrity Constraint Violation Messages* on page 26

Direct Loading of Data from Clients

SAP Sybase IQ supports bulk loading of remote data via the **LOAD TABLE USING CLIENT FILE** statement. **LOAD TABLE USING FILE** loads data on the local server, replacing the deprecated utility **iq_bcp**.

Note: The client and server must both be SAP Sybase IQ version 15.0 or later.

See also

- *Loads That Specify Input Data Format* on page 22
- *Considerations for Partitioned Table Loads* on page 25
- *Load and Insert Messages* on page 26
- *Integrity Constraint Violation Messages* on page 26

Considerations for Partitioned Table Loads

SAP Sybase IQ supports fully parallel bulk loads for range-, hash-, and hash-range partitioned tables.

Load performance for the same volume of data may vary depending on the type of the table being loaded. Unpartitioned tables load more quickly than partitioned tables. Range-partitioned tables load more quickly than hash- or hash-range partitioned tables. Loading data into a single range partition should be comparable to loading into an unpartitioned table. The load speed depends on a number of factors, including but not limited to the number of cores, bandwidth of the underlying I/O system, and amount of physical memory.

Load performance of partitioned tables also depends on partition-key data characteristics. Range-partitioned tables get best load performance when partition key data is grouped in partition order. Hash- and hash-range partitioned tables achieve best load performance when partition key data has uniform value distribution.

- The following applies to loading into a range-partitioned table or a hash-range partitioned table and the range-partitioning key column or the range subpartitioning key column:
  When you load data into a partitioned table, you can achieve the best performance when the partitioning column is placed first in the column list of the command. For a `LOAD` statement, list the partitioning columns before any other columns including large object (`LOB`) columns in the load file. If possible, use a preload process to rearrange data in the primary file. Similarly for an `INSERT...LOCATION` statement, list the partitioning columns before any other columns including large object (`LOB`) columns in the `SELECT` statement clause.

- The following applies to all partitioned tables and the partition key columns or subpartition key columns. Attempting to update the contents of a partitioning column returns this error:

```
"Updating partition key column on a partitioned table is not
allowed."
(SQLCODE -1009417L, SQLSTATE QCB15, Sybase error code 21055)
```

See also

- *Loads That Specify Input Data Format* on page 22
- *Direct Loading of Data from Clients* on page 25
- *Load and Insert Messages* on page 26
- *Integrity Constraint Violation Messages* on page 26

Load and Insert Messages

You can use a database option and a server startup switch to control insert and load messages.

You may see messages during insert and load operations. The `NOTIFY_MODULUS` database option adjusts the default frequency of notification messages during loads, or omits these messages. The `NOTIFY` option in the **LOAD** command overrides the `NOTIFY_MODULUS` setting.

The **IQMsgMaxSize** server property and the **-iqmsgsz** server startup switch control message log wrapping and the size of the message log file.

See also

- *Loads That Specify Input Data Format* on page 22
- *Direct Loading of Data from Clients* on page 25
- *Considerations for Partitioned Table Loads* on page 25
- *Integrity Constraint Violation Messages* on page 26

Integrity Constraint Violation Messages

LOAD TABLE allows you to control load behavior when integrity constraints are violated and to selectively log information about the violations.

- In Fast Projection (`FP`) indexes, continuous `NBit` dictionary compresion replaces `FP(1)`, `FP(2)`, and `FP(3)` byte dictionary compression. `FP(1)`, `FP(2)`, and `FP(3)` indexes roll over to `NBit(8)`,`NBit(16)`, and `NBit(24)` respectively. All data types except LOB (both character and binary) and BIT data types may be `NBit` columns.

  If `FP_NBIT_IQ15_COMPATIBILITY` is OFF, `IQ UNIQUE` determines whether the column loads as `Flat FP` or `NBit`. Setting `IQ UNIQUE` to 0 loads the column as `Flat`

FP. Columns without an IQ UNIQUE constraint load as NBit up to the NBit auto-sizing limits.

- New tiered HG index structure decouples load performance from HG index size. In 15.x, load throughput could degrade as the amount of data in an HG index increased. As the index grew, loading the same amount of data could take more time. The new tiered structure decouples load performance from the HG index size to increase throughput.

  The CREATE_HG_WITH_EXACT_DISTINCTS option determines whether newly created HG indexes are tiered or non-tiered. If this option is ON, all new HG indexes are non-tiered. To take advantage of the new structure, set this option to OFF. Use **sp_iqrebuildindex** to convert non-tiered HG indexes to tiered HG and vice-versa.

Using the **MESSAGE LOG ... ROW LOG** option with the **ONLY LOG** clause, you can direct the load to log information about specific types of integrity constraint violations both per violation in a message log file and per row in a row log file. If you do not specify the **ONLY LOG** clause, only the timestamps indicating the start and completion of the load are logged in these files.

The message log and row files for integrity constraint violations are distinct from the IQ message log file (.iqmsg).

You can specify whether to ignore UNIQUE, NULL, DATA VALUE, and FOREIGN KEY constraint violations that occur during a load and the maximum number of violations to ignore before initiating a rollback. You can also direct the load to log information about specific types of integrity constraint violations both per violation in a message log and per row in a row log.

See also

- *Loads That Specify Input Data Format* on page 22
- *Direct Loading of Data from Clients* on page 25
- *Considerations for Partitioned Table Loads* on page 25
- *Load and Insert Messages* on page 26

MESSAGE LOG Contents and Format

The **MESSAGE LOG** file contains row and column information for each integrity constraint violation logged.

A given load includes a timestamped header, row information, and a timestamped trailer. The header appears once per load. The trailer appears once if the statement executes successfully. The row information appears once for each integrity constraint violation logged.

The format of the header message is:

```
<datetime load started> Load Table <table-name>: Integrity
Constraint Violations
```

For example:

```
2009-05-24 23:04:31 Load Table Customers: Integrity Constraint
Violations
```

The row information message consists of:

- The row number within the table where this row would have been loaded, if an integrity constraint violation had not occurred.
- The type of integrity constraint violation detected.
- The column specified by the schema.

For example:

```
1267 DATA VALUE 4
3216 UNIQUE 1
3216 NULL 3
3216 NULL 6
9677 NULL 1
```

The format of the trailer message is:

```
<datetime load completed> Load Table <table-name> Completed
```

For example:

```
2009-05-24 23:05:43 LOAD TABLE Customers: Completed
```

Note: The number of rows (errors reported) in the **MESSAGE LOG** file may exceed the **IGNORE CONSTRAINT** option limit, because the load is performed by multiple threads running in parallel. More than one thread may report that the number of constraint violations has exceeded the specified limit.

See also

- *ROW LOG Contents and Format* on page 28
- *MESSAGE LOG and ROW LOG Example* on page 30

ROW LOG Contents and Format

The **ROW LOG** file contains row ID and data values for each row on which logged integrity constraint violations occurred.

The row data appears exactly once for a given row, regardless of the number of integrity constraint violations that occurred on that row. For a given load, there are three types of messages logged: a timestamped header, row data, and a timestamped trailer. The header appears once per load. The trailer appears once if the statement executes successfully.

The format of the header message is:

```
<datetime load started> Load Table <table-name>: Integrity
Constraint Violations
<formatting information>
```

where `<formatting information>` is the date, time, and datetime formats used in formatting the row data. For example:

```
2009-05-24 23:04:31 Load Table Customers: Integrity Constraint
Violations
Date Format: yyyy/mm/dd
Time Format: hh:mm:ss
Datetime format: yyyy/mm/dd hh:mm:ss
```

The row data message consists of:

- The row number within the table where this row would have been loaded, if an integrity constraint violation had not occurred.
- The data values in the row, separated by either a comma or the user-specified **LOG DELIMITED BY** separator.

For example:

```
3216 #Jones John#NULL#NULL#S#1945/01/12#NULL#
```

These rules determine the format of the data values in the row data message:

- When the data type is `VARBINARY` or `BINARY`, the data is represented by ASCII hexadecimal characters.
- `DATE` values are represented in the format specified by the DATE_FORMAT database option. The default format is YYYY-MM-DD.
- `DATETIME` and `TIMESTAMP` values are represented in the format specified by the TIMESTAMP_FORMAT database option. The default is YYYY-MM-DD HH:NN:SS.SSS.
- `TIME` values are represented in the format specified by the TIME_FORMAT database option. The default is HH:NN:SS.SSS.
- NULL values are represented by the token NULL.

Note: Filler fields do not appear in the row data message.

The format of the trailer message is:

```
<datetime load completed> Load Table <table-name>: Completed
```

For example:

```
2009-05-24 23:05:43 Load Table Customers: Completed
```

Note: The number of distinct errors in the **MESSAGE LOG** file may not exactly match the number of rows in the **ROW LOG** file. The difference in the number of rows is due to the parallel processing of the load performed by multiple threads. More than one thread may report that the number of constraint violations has exceeded the specified limit.

See also

- *MESSAGE LOG Contents and Format* on page 27
- *MESSAGE LOG and ROW LOG Example* on page 30

MESSAGE LOG and ROW LOG Example

An illustration of the contents and format of the **MESSAGE LOG** and **ROW LOG** files.

This statement creates a table to be loaded:

```
CREATE TABLE Customers(name VARCHAR(80) NOT NULL,
age TINYINT NULL,
sex CHAR(1) NOT NULL,
marital_status CHAR(1) NULL,
birthdate DATE NOT NULL,
credit_card VARCHAR(20)NOT NULL)
```

This statement loads the data into the `Customers` table:

```
LOAD TABLE Customers ...
IGNORE CONSTRAINT UNIQUE 200
MESSAGE LOG 'msg.log' ROW LOG 'row.log'
ONLY LOG UNIQUE, NULL, DATA VALUE
LOG DELIMITED BY '#'
```

The raw data is loaded from a disk file:

```
Jones John, 19, M, S, 06/19/83, CC
Cleven Bill, 56, M, OSIDJFJ, 02/23/43, CC
Jones John, 339, M, NULL, 01/12/45, NULL
NULL, 55, F, M, 10/02/37, ST
```

After the **LOAD TABLE** completes, the **MESSAGE LOG** file `msg.log` looks similar to:

```
2009-05-24 23:04:31 LOAD TABLE Customers: Integrity Constraint
Violations
1267 DATA VALUE 4
3216 UNIQUE 1
3216 NULL 6
9677 NULL 1
2009-05-24 23:05:43 LOAD TABLE Customers Completed
```

The **ROW LOG** file `row.log` looks similar to:

```
2009-05-24 23:04:31 LOAD TABLE Customers Integrity Constraint
Violations
Date Format: yyyy/mm/dd
Time Format: hh:mm:ss
Datetime format: yyyy/mm/dd hh:mm:ss

1137 #Jones John#19#M#S#1983/06/19#CC#
1267 #Cleven Bill#56#M#OSIDJFJ#1943/02/23#CC#
3216 #Jones John#NULL#NULL#S#1945/01/12#NULL#
9677 #NULL#55#F#M#1937/10/02#ST#

2009-05-24 23:05:43 LOAD TABLE Customers Completed
```

See also

- *MESSAGE LOG Contents and Format* on page 27

- *ROW LOG Contents and Format* on page 28

Binary Load Formats

For fast data loading into SAP Sybase IQ, create data files in binary format, then load the data using the **FORMAT BINARY** and **BINARY** column specification clauses of **LOAD TABLE**.

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

- `DATE`
- `TIME`
- `DATETIME`
- `NUMERIC`

Binary Load Format and Load Efficiency

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

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

The IQ binary load format is a fixed-width load. The load can determine the width of each column and length of each row from information in the table definition.

Note: Binary load format is endian-sensitive, utilizing native binary data types to represent data.

See also

- *Operating System Native Data Types* on page 34
- *DATE* on page 34
- *TIME* on page 35
- *TIMESTAMP* on page 35
- *NUMERIC and DECIMAL* on page 36
- *NULL Value Loads* on page 37

Operating System Native Data Types

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

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

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

See also

- *Binary Load Format and Load Efficiency* on page 33
- *DATE* on page 34
- *TIME* on page 35
- *TIMESTAMP* on page 35
- *NUMERIC and DECIMAL* on page 36
- *NULL Value Loads* on page 37

DATE

`DATE` column data is stored in SAP Sybase IQ as 4 bytes (a 32-bit unsigned integer) representing the number of days since 0000-01-01.

To convert a calendar date to the SAP Sybase IQ binary format, for a given year, month, and day, use:

```
year = current_year - 1;
days_in_year_0000 = 366;
binaryDateValue = (year * 365)
+ (year / 4)
- (year / 100)
+ (year / 400)
+ days_in_year_0000
```

```
+ day_of_current_year
-1;
```

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

See also

- *Binary Load Format and Load Efficiency* on page 33
- *Operating System Native Data Types* on page 34
- *TIME* on page 35
- *TIMESTAMP* on page 35
- *NUMERIC and DECIMAL* on page 36
- *NULL Value Loads* on page 37

TIME

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

Compute the microsecond quantity for a given hour, minute, second, and microsecond (*usec*):

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

See also

- *Binary Load Format and Load Efficiency* on page 33
- *Operating System Native Data Types* on page 34
- *DATE* on page 34
- *TIMESTAMP* on page 35
- *NUMERIC and DECIMAL* on page 36
- *NULL Value Loads* on page 37

TIMESTAMP

`TIMESTAMP` data is stored as a 64-bit unsigned integer and represents a quantity in microseconds.

You can compute a binary `TIMESTAMP` value for a given year, month, day, hour, minute, second, and microsecond as follows:

Compute *binaryDateValue* for the date.

Compute *binaryTimeValue* for the time.

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

See also

- *Binary Load Format and Load Efficiency* on page 33
- *Operating System Native Data Types* on page 34
- *DATE* on page 34
- *TIME* on page 35
- *NUMERIC and DECIMAL* on page 36
- *NULL Value Loads* on page 37

NUMERIC and DECIMAL

Formats for `NUMERIC` and `DECIMAL` data types vary as a function of precision.

The value must be right-padded with zeros to the full scale of the value. The value must also be fully left-padded with zeros, but with binary programming, padding happens automatically. Once the values are padded, the decimal point is removed. For example, 12.34 looks like:

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

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

- If precision <= 4, binary format is identical to native operating system binary format for 2-byte integer quantity.
- If precision is between 5 and 9, binary format is identical to native operating system binary format for a 4-byte integer quantity.
- If precision is between 10 and 18, binary format is identical to native operating system binary format for an 8-byte integer quantity.
- If precision >= 19, there is a special format that uses this C++ struct definition:

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

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

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

```
erracc = 0
exp = 0
```

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

- `lower order digit = digit[i] & 0x00FF`
- `high order digit = digit[i] & 0xFF00`

For example, consider the value 100 represented as `NUMERIC`(20). The binary layout of this value is:

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

Sign = 0x01
Number digits = 0x01
Exponent = 0x50
Erracc = 0x00
Digits = 0x0064
```

As another example, consider the value 32769:

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

Sign = 0x01
Number digits = 0x02
Exponent = 0x50
Erracc = 0x00
Digits = 0x0ad1 0x0003
```

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

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

See also

- *Binary Load Format and Load Efficiency* on page 33
- *Operating System Native Data Types* on page 34
- *DATE* on page 34
- *TIME* on page 35
- *TIMESTAMP* on page 35
- *NULL Value Loads* on page 37

NULL Value Loads

The most expedient way to insert `NULL` values is to use the `NULL` byte in the input file and specify **WITH NULL BYTE** in the column specification of the **LOAD TABLE** statement.

This is done by terminating each data field in the input file with “x00” or “x01”.

Terminating a data field in the input file with “x01” instructs the load to insert NULL into the column. For example:

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

If the content of the load input file is 000b32cb00000b32cc00, two rows are loaded to the table. The first row is May 7, 2009 and the second May 8, 2009. A NULL byte is added to the input file after each binary date. If you want NULL loaded into the first row, change the value of the NULL byte in the input file to “x01”.

```
000b32cb01000b32cc00
```

As another example, to load the value 32769 into a `NUMERIC`(20) column, the input file contains:

```
0x0102 0x5000 0x0ad1 0x0003 0x0000 0x00
```

This includes the NULL byte.

To load 23456789012345678.12 into a column defined as `NUMERIC`(19,2), the load input file contains:

```
0x0106 0x4f00 0x04b0 0x162e 0x04d2 0x1ed2 0x0d80 0x0002 0x0000 0x00
```

The digits are followed by the NULL BYTE (0x00).

There are seven (numbered 0 – 6) unsigned shorts in the digits array of the structure that represents this numeric quantity. “digits[0]” contains the least-significant digits.

```
digits[0] = 0x04b0 (decimal 120)
digits[1] = 0x162e (decimal 5678)
digits[2] = 0x04d2 (decimal 1234)
digits[3] = 0x1ed2 (decimal 7890)
digits[4] = 0x0d80 (decimal 3456)
digits[5] = 0x0002 (decimal 2)
digits[6] = 0x0000
```

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

ZEROS is interpreted as follows:

- The column is set to NULL if the input data is entirely binary zeros (as opposed to character zeros).
- If the input data is character zero:
  - NULL(ZEROS) never causes the column to be NULL.
  - NULL('0') causes the column to be NULL. For example, load:

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

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

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

Execute:

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

The results:

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

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

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

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

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

Execute:

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

The results:

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

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

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

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

SELECT * FROM T1;
```

```
c1      c2
NULL    1234
NULL    1234
```

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

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

See also

- *Binary Load Format and Load Efficiency* on page 33
- *Operating System Native Data Types* on page 34
- *DATE* on page 34
- *TIME* on page 35
- *TIMESTAMP* on page 35
- *NUMERIC and DECIMAL* on page 36

Using the INSERT Statement

The **INSERT** statement allows you to insert data without first putting it into a flat file.

Using this command, you can either:

- Insert a specified set of values row by row
- Insert directly from database tables

Inserting Specified Values Row by Row

To add specified values to a table row by row, use Syntax 1 for the **INSERT** statement. SAP Sybase IQ inserts the first value you specify into the first column you specify, the second value you specify into the second column, and so on.

If you omit the list of column names, the values are inserted into the table columns in the order in which the columns were created (the same order as **SELECT *** retrieves). SAP Sybase IQ inserts the row into the table wherever room is available.

Values can be NULL, any positive or negative number, or a literal.

- Enclose values for `CHAR`, `VARCHAR`, `DATE`, `TIME`, and `TIMESTAMP` or `DATETIME` columns in single or double quotation marks. To indicate a value with a quotation in it, use a different set of quotes for the outer quote, such as “Smith' s”.
- For `DATE`, `TIME`, and `TIMESTAMP` or `DATETIME` columns, you must use a specific format.

Note: The `TIMESTAMP` and `DATETIME` data types are identical.

When you specify values for only some of the columns in a row, NULL is inserted for columns with no value specified, if the column allows NULL.

If you specify a NULL value, the destination column must allow NULLs, or the INSERT is rejected and an error message is produced in the message log. By default, SAP Sybase IQ columns allow NULLs, but you can alter this by specifying **NOT NULL** on the column definition in the **CREATE TABLE** statement, or in other ways, such as using a primary key, for example.

The following example adds 1995-06-09 into the `l_shipdate` column and 123 into the `l_orderkey` column in the `lineitem` table.

```
INSERT INTO lineitem
    (l_shipdate, l_orderkey)
VALUES('1995-06-09', 123)
```

If you are inserting more than a small number of data rows, it is more efficient to insert selected rows directly from a database, or to load data from a flat file with the **LOAD TABLE** statement,

than to insert values row by row. Consider using a select statement with a few unions instead of inserting values for a few rows, because this requires only a single trip to the server.

You can use **INSERT VALUES** to support multiple rows.

For example:

```
INSERT INTO lineitem(l_shipdate, l_orderkey)
VALUES ('1995-06-09', 123),
'2001-03-28', 300),
('2010-04-01', 413);
```

See also

- *Inserting Selected Rows from the Database* on page 42

Inserting Selected Rows from the Database

You can insert any number of rows of data, based on the results of a general **SELECT** statement.

To insert data from other tables in the current database, or from a database that is defined as a specialty data store to SAP Sybase IQ, use the **INSERT** statement.

For maximum efficiency, insert as many rows as possible in one **INSERT** statement. To insert additional sets of rows after the first insert, use additional **INSERT** statements.

Like other SQL databases, SAP Sybase IQ inserts data by matching the order in which columns are specified in the destination column list and the select list; that is, data from the first column in the select list is inserted into the first destination column, and so on. For both **INSERT SELECT** and **INSERT VALUES**, if you omit destination column names, SAP Sybase IQ inserts data into columns in the order in which they were created.

The tables you are inserting into must exist in the database you are currently connected to. SAP Sybase IQ inserts the data into all indexes for the destination columns.

The columns in the table in the select list and in the table must have the same or compatible data types. In other words, the selection's value must be, or must be able to be converted to, the data type of the table's column.

With this form of the **INSERT** statement you can specify any of the insert-load-options.

Example

This example shows an insert from one table, `partsupp`, to another, `lineitem`, within the same database.

The data from the source column `l_quantity` is inserted into the destination column `ps_availqty`.

```
INSERT INTO partsupp(ps_availqty)
SELECT l_quantity FROM lineitem
```

See also

- *Inserting Specified Values Row by Row* on page 41

Inserting from a Different Database

You can insert data from tables in any accessible database.

- Tables in either the IQ store or the catalog store of the database you are currently connected to.
- Tables in an Adaptive Server® Enterprise database.
- A *proxy table* in your current database, that corresponds to a table in a database on a remote server.

Inserting Data Directly from Adaptive Server Enterprise

To insert data from an Adaptive Server Enterprise (ASE) or SQL Server database, use the **LOCATION** syntax of the **INSERT** statement.

To use insert data directly from SAP Sybase IQ, all of the following must be true:

- The SAP Sybase connectivity libraries must be installed on your system, and the load library path environment variable for your platform must point to them.
- The Adaptive Server Enterprise server to which you are connecting must exist in the `interfaces` file on the local machine.
- You must have read permission on the source ASE or SAP Sybase IQ database, and INSERT permission on the target SAP Sybase IQdatabase.

1. Connect to both the Adaptive Server Enterprise and the SAP Sybase IQ database using the same user ID and password.
2. On the SAP Sybase IQ database, issue:

```
INSERT INTO iq_table
LOCATION 'ase_servername.ase_dbname'
{ SELECT col1, col2, col3,...
FROM owner.ase_table }
```

3. Issue a **COMMIT** to commit the insert.

When SAP Sybase IQ connects to the remote server, **INSERT...LOCATION** can also use the remote login for the user ID of the current connection, if a remote login has been created with **CREATE EXTERNLOGIN** and the remote server has been defined with a **CREATE SERVER** statement.

Note: You can also use this method to move selected columns between two SAP Sybase IQ databases.

Loading ASE Text and Images

SAP Sybase IQdoes not support the Adaptive Server Enterprise data type `TEXT`, but you can execute **INSERT...LOCATION** (Syntax 3) from both an IQ `CHAR` or `VARCHAR` column with length is greater than 255 bytes, or a `LONG VARCHAR` column, and from an ASE database column of data type `TEXT`. ASE `TEXT` and `IMAGE` columns can be inserted into columns of

other SAP Sybase IQ data types, if SAP Sybase IQ supports the internal conversion. **INSERT...LOCATION** does not support the use of variables in the **SELECT** statement. By default, if a remote data column contains over 2GB, SAP Sybase IQ silently truncates the column value to 2GB.

Users must be specifically licensed to use the Unstructured Data Analytics functionality.

You may substitute curly braces {} for the single quotation marks that delimit the SELECT statement. (However, curly braces represent the start and end of an escape sequence in the ODBC standard, and may generate errors in the context of ODBC.)

Example

The following command inserts data from the `l_shipdate` and `l_orderkey` columns of the `lineitem` table from the SAP Sybase IQ database `iq11db.dba` on the server `detroit`, into the corresponding columns of the `lineitem` table in the current database:

```
INSERT INTO lineitem
    (l_shipdate, l_orderkey)
     LOCATION 'detroit.iq11db'
    { SELECT l_shipdate, l_orderkey
    FROM lineitem }
```

- The destination and source columns may have different names.
- The order in which you specify the columns is important, because data from the first source column named is inserted into the first target column named, and so on.
- You can use the predicates of the **SELECT** statement within the **INSERT** command to insert data from only certain rows in the table.

Example

This example inserts the same columns as the previous example, but only for the rows where the value of `l_orderkey` is 1. Also in this example, the TDS packet size is specified as 512 bytes.

```
INSERT INTO lineitem
    (l_shipdate, l_orderkey)
    LOCATION 'detroit.iqdb'
    PACKETSIZE 512
    { SELECT l_shipdate, l_orderkey
    FROM lineitem
    WHERE l_orderkey = 1 }
```

Interactive Data Imports

If you are inserting small quantities of data, you may prefer to enter it interactively through Interactive SQL, using the **INSERT** statement.

For example, you can insert listed values a single row at a time with the following command:

```
INSERT INTO T1
VALUES ( ... )
```

Note: Do not use the Import option on the Interactive SQL Data menu. It is not supported for use with SAP Sybase IQ databases.

Moving Data Between Systems with Different Endian Formats

You can move data from a database in big-endian format to a database in little-endian format.

Prerequisites

Note: Before you begin, make sure that you have a process for capturing your database and table schema.

The following example loads a table named `lineitem` and identifies one extract file on UFS (file system) called `lineitem_binary.inp`.

Check operating system documentation for the maximum file size for your system. For example, an extract file on Sun Solaris x64 has a maximum size of 512GB.

Task

This procedure moves table definitions but does not include migration of database objects, such as stored procedures or events, which you must re-create.

For example, SAP Sybase IQ databases built on Sun64 SPARC systems store binary data in big-endian (most significant byte first) format. Because Sun Solaris x64 is a little-endian system, you cannot upgrade SAP Sybase IQ databases built on Sun64 SPARC with **ALTER DATABASE UPGRADE** to run on Sun Solaris x64 systems.

To move data for each database across hardware platforms of different endian structures, you must:

- Copy the database schema from the source platform (tables, indexes, and so on).
- Create a new database on the target platform.
- Perform a binary data dump from the source database.
- Load data into the new target database.

1. Activate the extract utility:

```
SET TEMPORARY OPTION Temp_Extract_Name1 =
'lineitem_binary.inp'
```

```
SET TEMPORARY OPTION Temp_Extract_Name2 = ''
```

2. Set up a binary extract of the `lineitem` table:

```
SET TEMPORARY OPTION Temp_Extract_Binary = 'on'
```

```
SET TEMPORARY OPTION Temp_Extract_Swap = 'off'
```

3. Place output in the file `lineitem_binary.inp`:

```
SELECT * FROM lineitem
```

4. Turn off the extract utility:

```
SET TEMPORARY OPTION Temp_Extract_Name1 = ''
```

5. Create a duplicate of your database on the target system.
6. Assuming table `lineitem` as defined below, load the `lineitem` table as follows:

```
LOAD TABLE lineitem
( l_orderkey     BINARY WITH NULL BYTE,
l_partkey        BINARY WITH NULL BYTE,
l_suppkey        BINARY WITH NULL BYTE,
l_linenumber     BINARY WITH NULL BYTE,
l_quantity       BINARY WITH NULL BYTE,
l_extendedprice  BINARY WITH NULL BYTE,
l_discount       BINARY WITH NULL BYTE,
l_tax            BINARY WITH NULL BYTE,
l_returnflag     BINARY WITH NULL BYTE,
l_linestatus     BINARY WITH NULL BYTE,
l_shipdate       BINARY WITH NULL BYTE,
l_commitdate     BINARY WITH NULL BYTE,
l_receiptdate    BINARY WITH NULL BYTE,
l_shipinstruct   BINARY WITH NULL BYTE,
l_shipmode       BINARY WITH NULL BYTE,
l_comment        BINARY WITH NULL BYTE )
FROM 'C:\\mydata\\lineitem_binary.inp'
FORMAT BINARY
STRIP OFF
QUOTES OFF
ESCAPES OFF
PREVIEW ON
BYTE ORDER HIGH;
COMMIT
```

   Note particularly two clauses:
   - BINARY WITH NULL BYTE is required when loading a binary file.
   - BYTE ORDER HIGH specifies the byte order from the system where the data *originated*. The source database in this example is a big-endian platform; therefore, this data requires byte order HIGH. (Little-endian databases require byte order LOW.)

When loading a multiplex database, use absolute (fully qualified) paths in all file names. Do not use relative path names.

Insertions into Primary and Foreign Key Columns

You load or insert data into primary key and foreign key columns just as you would into any other column.

When you insert into a primary key, SAP Sybase IQ checks that each value is unique. If it is not, an error occurs.

Load or Extraction of Large Object Data

Loads and extractions of large object data relate to unstructured data analytics. You must be specifically licensed to use the unstructured data analytics functionality.

Data Conversion on Insertion

The data you enter into the SAP Sybase IQ database may come from diverse sources.

Not all of your data will match the SAP Sybase IQ data types exactly. Some of it will need to be converted. Data is converted in two ways: explicitly and implicitly. For example, to insert `INT` data into a `CHAR` column you must convert it explicitly.

Implicit conversions can occur when you:

- Insert data selected from another column in the same database
- Insert data selected from another database
- Load data from a flat file

When an explicit conversion is needed, the way you specify the conversion depends on whether you are loading from a flat file or inserting selected rows:

- In the **LOAD TABLE** statement, convert data explicitly by specifying a format in the *column-spec*.
- In the **INSERT** statement, convert data explicitly using the data conversion functions **CAST**, **CONVERT**, and **DATEPART** in the **SELECT** statement or **VALUES** list.

While most SAP Sybase IQ data types are fully compatible with SAP® Sybase SQL Anywhere® and Adaptive Server Enterprise data types of the same name, there are some differences.

For compatibility among versions, a few data types have been defined as synonyms of other data types:

- `DECIMAL` is a synonym for `NUMERIC`.
- `INTEGER` is a synonym for `INT`.
- `DATETIME` is a synonym for `TIMESTAMP`.
- `FLOAT` (*precision*) is a synonym for `REAL` or `DOUBLE`, depending on the value of *precision*. For Adaptive Server Enterprise, `REAL` is used for *precision* less than or equal to 15, and `DOUBLE` for *precision* greater than 15. For SAP Sybase IQ and SAP SQL Anywhere, the cutoff is platform-dependent, but on all platforms the cutoff value is greater than 22.
- `MONEY` is an Adaptive Server Enterprise-compatible synonym for `NUMERIC`(19,4), allowing NULL.
- `SMALLMONEY` is an Adaptive Server Enterprise-compatible synonym for `NUMERIC`(10,4), allowing NULL.

You can use a synonym interchangeably with its standard data type. Data is stored internally as the standard data type, where synonyms exist. In error messages, the standard name appears in place of the synonym.

Note: By default, SAP Sybase IQ assumes that input data is binary (numeric data) and tries to insert it that way. However, this presumes that the input column length in bytes must match the destination column length in bytes. If not, the insert fails or leads to unexpected results. For example, if you attempt to insert an input column with integer data of 4 bytes into a `SMALLINT` destination column, SAP Sybase IQ loads only the first 2 bytes of that input column.

Load Conversion Options

There are several conversion options for loading from flat files using the **LOAD TABLE** statement.

Option	SAP Sybase IQ Data types	Action
ASCII	TINYINT, SMALLINT, INT (or INTEGER), UNSIGNED INT, BIGINT, UNSIGNED BIGINT, NUMERIC (or DECIMAL), REAL, DOUBLE, BIT, DATE, TIME, TIMESTAMP (or DATETIME)	By default, SAP Sybase IQ assumes input data is binary of appropriate width for the data type. Using ASCII allows you to tell SAP Sybase IQ that data is in character format and lets you specify how wide it is. This option allows E notation for REAL data, but it might degrade performance. If a problem occurs when converting these data types to CHAR or VARCHAR, SAP Sybase IQ logs the failure as an error or warning in the .iqmsg file. If the CONVERSION_ERROR option is ON, SAP Sybase IQ reports the problem as an error. If the CONVERSION_ERROR option is OFF, the problem is reported as a warning
ASCII	CHAR, VARCHAR	By default, SAP Sybase IQ assumes the same column width between source and destination columns, which may cause it to incorrectly read the input file. This option lets you specify a different width for the input column.
DATE	DATE	Converts ASCII date input of a fixed format to binary.
DATETIME	TIMESTAMP (or DATETIME) or TIME	Converts ASCII time or date/time input of a fixed format to binary. The input specification is based on either a 12-hour or 24-hour clock.
TIME	TIME	Converts ASCII time input of a fixed format to binary.
NULL	all	Lets you specify which input data values to convert to NULL on insert.

Note: When loading from a flat file, use binary data if you have a choice of using binary or character data. Using binary input may improve performance by eliminating conversion costs.

See also

- *Explicit Data Conversions* on page 55

Explicit Data Conversions

When you use the **INSERT** statement to insert data directly from a database rather than from a flat file, you cannot use the load conversion options.

If the data requires explicit conversion, you must use one of the conversion functions, **CAST** or **CONVERT**, in the **SELECT** statement or **VALUES** list where you specify the data to be inserted. If the data is converted implicitly, SAP Sybase IQ handles the conversion automatically.

An implicit or explicit conversion is required whenever data types in a **SELECT** statement need to match, but do not. This occurs when you execute an **INSERT SELECT** from one data type to another, but it also occurs whenever you compare or compute values for differing data types.

These conversions apply to data within a SAP Sybase IQ database, or coming from an SAP SQL Anywhere database, or any other database that is connected as a specialty data store.

Conversions (implicit (I), explicit (E), and unsupported (U) conversions) when there is no **WHERE** clause in the **SELECT** statement, or when the **WHERE** clause is based on a comparison operation (=, > or <) include:

	To:																
From:	**ti**	**si**	**in**	**ui**	**bi**	**ub**	**nu**	**rl**	**dl**	**bt**	**dt**	**tm**	**ts**	**ch**	**vc**	**bn**	**vb**
tinyint	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I
smallint	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I
int	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I
unsigned int	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I
bigint	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I
unsigned bigint	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I
numeric	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	U	U

	To:																
From:	ti	si	in	ui	bi	ub	nu	rl	dl	bt	dt	tm	ts	ch	vc	bn	vb
real	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	U	U
double	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	U	U
bit	I	I	I	I	I	I	I	I	I	I	U	U	U	I	I	I	I
date	E	E	E	E	E	E	E	E	E	U	I	U	I	E	E	U	U
time	E	E	E	E	E	E	E	E	E	U	U	I	E	E	E	U	U
time-stamp	E	E	E	E	E	E	E	E	E	U	E	I	I	E	E	U	U
char	E	E	E	E	E	E	E	E	E	I	E	E	E	I	I	I	I
varchar	E	E	E	E	E	E	E	E	E	I	E	E	E	I	I	I	I
binary	I	I	I	I	I	I	U	U	U	U	U	U	U	I	I	I	I
varbinary	I	I	I	I	I	I	U	U	U	U	U	U	U	I	I	I	I

The descriptions of the codes used in the tables include:

Code	Data Type	Code	Data Type	Code	Data Type
ti	tinyint	nu	numeric	ts	timestamp
si	smallint	rl	real	ch	char
in	int	dl	double	vc	varchar
ui	unsigned int	bt	bit	bn	binary
bi	bigint	dt	date	vb	varbinary
ub	unsigned bigint	tm	time		

Conversions when the **WHERE** clause in a **SELECT** statement is based on an arithmetic operation (+, –, and so on) include:

Table 1. SAP Sybase IQ Conversions for Arithmetic Operations

	To:																
From:	ti	si	in	ui	bi	ub	nu	rl	dl	bt	dt	tm	ts	ch	vc	bn	vb
tinyint	I	I	I	I	I	I	I	I	I	I	U	U	U	E	E	I	I

	To:																
From:	**ti**	**si**	**in**	**ui**	**bi**	**ub**	**nu**	**rl**	**dl**	**bt**	**dt**	**tm**	**ts**	**ch**	**vc**	**bn**	**vb**
smallint	I	I	I	I	I	I	I	I	I	I	U	U	U	E	E	I	I
int	I	I	I	I	I	I	I	I	I	I	U	U	U	E	E	I	I
unsigned int	I	I	I	I	I	I	I	I	I	I	U	U	U	E	E	I	I
bigint	I	I	I	I	I	I	I	I	I	I	U	U	U	E	E	I	I
unsigned bigint	I	I	I	I	I	I	I	I	I	I	U	U	U	E	E	I	I
numeric	I	I	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
real	I	I	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
double	I	I	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
bit	I	I	I	I	I	I	I	I	I	I	U	U	U	I	I	I	I
date	U	U	U	U	U	U	U	U	U	U	U	I	U	U	U	U	U
time	U	U	U	U	U	U	U	U	U	U	I	U	U	U	U	U	U
time-stamp	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
char	E	E	E	E	E	E	E	E	E	I	U	U	U	I	I	I	I
varchar	E	E	E	E	E	E	E	E	E	I	U	U	U	I	I	I	I
binary	I	I	I	I	I	I	U	U	U	U	U	U	U	I	I	I	I
varbinary	I	I	I	I	I	I	U	U	U	U	U	U	U	I	I	I	I

Note: In arithmetic operations, `bit` data is implicitly converted to `tinyint`.

Conversions for the **INSERT** and **UPDATE** statements include:

	To:																
From:	**ti**	**si**	**in**	**ui**	**bi**	**ub**	**nu**	**rl**	**dl**	**bt**	**dt**	**tm**	**ts**	**ch**	**vc**	**bn**	**vb**
tinyint	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I
smallint	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I
int	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I

	To:																
From:	**ti**	**si**	**in**	**ui**	**bi**	**ub**	**nu**	**rl**	**dl**	**bt**	**dt**	**tm**	**ts**	**ch**	**vc**	**bn**	**vb**
unsigned int	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I
bigint	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I
unsigned bigint	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	I	I
numeric	I	I	I	I	I	I	I	I	I	E	E	E	E	E	E	U	U
real	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	U	U
double	I	I	I	I	I	I	I	I	I	I	E	E	E	E	E	U	U
bit	I	I	I	I	I	I	I	I	I	I	U	U	U	I	I	I	I
date	E	E	E	E	E	E	E	E	E	E	I	U	I	E	E	U	U
time	E	E	E	E	E	E	E	E	E	E	U	I	E	E	E	U	U
time-stamp	E	E	E	E	E	E	E	E	E	E	E	I	I	E	E	U	U
char	I	I	I	I	I	I	I	I	I	I	E	E	E	I	I	I	I
varchar	I	I	I	I	I	I	I	I	I	I	E	E	E	I	I	I	I
binary	I	I	I	I	I	I	U	U	U	I	U	U	U	I	I	I	I
varbinary	I	I	I	I	I	I	U	U	U	I	U	U	U	I	I	I	I

See also

- *Load Conversion Options* on page 54
- *Column Width Issues* on page 59
- *Faster Date and Time Loads* on page 60
- *ASCII Input Conversion* on page 61
- *The DATE Option* on page 63
- *The DATETIME Conversion Option* on page 65
- *NULL Data Conversions* on page 68

Column Width Issues

SAP Sybase IQ assumes the width of the input data is the same as the destination column width and reads the input file accordingly.

If they are not the same width, SAP Sybase IQ may read too few or too many bytes of the input file for that column. The result is that the read for that column may be incorrect, and the reads for subsequent columns in the input file will also be incorrect, because they will not start at the correct position in the input file.

For example, if `input_column1` is 15 bytes wide and `destination_column1` is 10 bytes wide, and you do not specify the **ASCII** conversion option, SAP Sybase IQ assumes the input column is only 10 bytes wide. This is fine for `destination_column1`, because the input data is truncated to 10 bytes. But it also means that SAP Sybase IQ assumes that the next column in the input file starts at byte 11, which is still in the middle of the first column, instead of at byte 16, which is the correct starting position of the next column.

Conversely, if `input_column1` is 10 bytes wide and `destination_column1` is 15 bytes wide, and you do not specify the **ASCII** conversion option, SAP Sybase IQ assumes the input column is 15 bytes wide. This means that SAP Sybase IQ reads all of `input_column1` plus 5 bytes into the next column in the input file and inserts this value into `destination_column1`. So, the value inserts into `destination_column1` and all subsequent columns are incorrect.

To prevent such problems, use the **ASCII** conversion option. With this option, SAP Sybase IQ provides several ways to specify the fixed or variable width of an input column. Your input data can contain fixed-width input columns with a specific size in bytes, variable width input columns with column delimiters, and variable width input columns defined by binary prefix bytes.

See also

- *Load Conversion Options* on page 54
- *Explicit Data Conversions* on page 55
- *Faster Date and Time Loads* on page 60
- *ASCII Input Conversion* on page 61
- *The DATE Option* on page 63
- *The DATETIME Conversion Option* on page 65
- *NULL Data Conversions* on page 68

Faster Date and Time Loads

SAP Sybase IQ has performance optimizations built in for ASCII-to-binary conversions on date, time, and datetime data during loads. If the raw data you are loading exactly matches one of these formats, you can significantly decrease load time by using the appropriate format.

The recognized formats are:

- "YYYY-MM-DD"
- "YYYY/MM/DD"
- "YYYY.MM.DD"
- "YYYYMMDD"
- "MM-DD-YYYY"
- "MM/DD/YYYY"
- "DD-MM-YYYY"
- "DD/MM/YYYY"
- "DD.MM.YYYY"
- "HH:NN:SS"
- "HHNNSS"
- "HH:NN:SS.S"
- "HH:NN:SS.SS"
- "HH:NN:SS.SSS"
- "HH:NN:SS.SSSS"
- "HH:NN:SS.SSSSS"
- "HH:NN:SS.SSSSSS"
- "YYYY-MM-DD HH:NN:SS"
- "YYYYMMDD HHNNSS"
- "YYYY-MM-DD HH:NN:SS.S"
- "YYYY-MM-DD HH:NN:SS.SS"
- "YYYY-MM-DD HH:NN:SS.SSS"
- "YYYY-MM-DD HH:NN:SS.SSSS"
- "YYYY-MM-DD HH:NN:SS.SSSSS"
- "YYYY-MM-DD HH:NN:SS.SSSSSS"

When you load a table with one or more date, time, or datetime columns and the input format is in one of the above formats, the load can run significantly faster if you explicitly specify the appropriate format on the load statement. Otherwise, the load can run very slowly.

Suppose that your table had a date column, created as follows:

```
CREATE TABLE table1(c1 DATE);
```

To load the table, use a statement like this:

```
LOAD TABLE table1 (c1 ASCII(10)) FROM ...
```

If the raw data format is in a format that has been optimized (such as YYYY-MM-DD), the load will be much faster.

See also

- *Load Conversion Options* on page 54
- *Explicit Data Conversions* on page 55
- *Column Width Issues* on page 59
- *ASCII Input Conversion* on page 61
- *The DATE Option* on page 63
- *The DATETIME Conversion Option* on page 65
- *NULL Data Conversions* on page 68

ASCII Input Conversion

Convert ASCII input data to binary.

Use the **ASCII** conversion option to either:

- Convert ASCII input data to binary and specify the width of the input column so data can be read in correctly for that column, or
- Insert ASCII data into an `ASCII` data type column when the width of the input column is different from the width of the destination column. This option lets you specify how much of the input data it should read for each column.

You can use this option with any of the SAP Sybase IQ data types, with 1, 2, or 4 prefix bytes, and with a column delimiter.

Truncation of Data for VARCHAR and CHAR Columns

If the width of the input column is greater than the width of the destination column, SAP Sybase IQ truncates the data upon insertion.

If the width of the input data is less than the width of the destination column, for `CHAR` or `VARCHAR` data types SAP Sybase IQ pads the data with spaces in the table upon insertion.

Variable width inserts to a `VARCHAR` column will not have trailing blanks trimmed, while fixed-width inserts to a `VARCHAR` column will be trimmed. For example, assume that you are inserting into column `varcolumn` in a table called `vartable`. The following would constitute a fixed-width insert, where the value would not be trimmed because you explicitly say to include the two blanks (indicated by __ here):

```
INSERT INTO vartable VALUES ('box__')
```

If instead you inserted the same value from a flat file using delimited input, it would be a variable-width insert, and the trailing blanks would be trimmed.

The **ASCII** conversion option works with the SAP Sybase IQ data types. The example inserts the data from the flat ASCII file `shipinfo.t` into the SAP Sybase IQ table `lineitem` and summarizes the content and format of the input data and the table.

File shipinfo.t			Table lineitem		
Column	**Format**	**Width**	**Column**	**Data Type**	**Width**
l_shipmode	CHAR	15	l_shipmode	VARCHAR	30
l_quantity	ASCII	8	l_quantity	INT	4

For the `l_shipmode` column, you insert ASCII data into an ASCII column (that has a `VARCHAR` data type). Notice the width of the two columns is different. In order for the insert on this column and the subsequent `l_quantity` column to be correct, you specify the width of the `l_shipmode` column so the correct amount of input data is read at the correct position.

For the `l_quantity` column, you are inserting ASCII data into a binary column (**INT** data type). In order for the insert on this column to be correct, you must convert the input data into binary and indicate the width of the input column.

The command for this is shown in the following UNIX example.

```
LOAD TABLE lineitem(
    l_shipmode ASCII(15),
    l_quantity ASCII(8),
FILLER(1))
FROM '/d1/MILL1/shipinfo.t'
PREVIEW ON
```

See also

- *Load Conversion Options* on page 54
- *Explicit Data Conversions* on page 55
- *Column Width Issues* on page 59
- *Faster Date and Time Loads* on page 60
- *The DATE Option* on page 63
- *The DATETIME Conversion Option* on page 65
- *NULL Data Conversions* on page 68

Substitution of NULL or Blank Characters

SAP Sybase IQ supports zero-length `VARCHAR` data.

If the length of a `VARCHAR` cell is zero, and the cell is not NULL, you get a zero-length cell if the option `NON_ANSI_NULL_VARCHAR=OFF`.

If `NON_ANSI_NULL_VARCHAR=ON`, a NULL is inserted.

For all other data types, if the length of the cell is zero, a NULL is inserted.

The DATE Option

Use the **DATE** conversion option to insert ASCII data that is stored in a fixed format into a `DATE` column.

This option converts the ASCII data input to binary and specifies the format of the input data. (The **DATE** format is used internally to interpret the input; it does not affect the storage or output format of the data.)

Example

In this Windows example, data for the `l_shipdate` column is converted from the specified format into binary. The 1-byte **FILLER** value skips over carriage returns in the input file.

```
LOAD TABLE lineitem(
    l_orderkey NULL(ZEROS) ASCII(4),
    l_partkey ASCII(3),
    l_shipdate DATE('MM/DD/YY'),
    l_suppkey ASCII(5),
FILLER(1))
FROM 'C:\\MILL1\\shipinfo.t'
PREVIEW ON
```

See also

- *Load Conversion Options* on page 54
- *Explicit Data Conversions* on page 55
- *Column Width Issues* on page 59
- *Faster Date and Time Loads* on page 60
- *ASCII Input Conversion* on page 61
- *The DATETIME Conversion Option* on page 65
- *NULL Data Conversions* on page 68

DATE Formats

Specify the format of the input data using y or Y for years, m or M for months, d or D for days, and j or J for Julian days.

The length of the format string is the width of the input column.

Option	Meaning
yyyy or YYYY yy or YY	Represents number of year. Default is 1900.

Option	Meaning
mm or MM	Represents number of month. Always use leading zeros for number of the month where appropriate, for example '05' for May. If you omit the month from a DATE value, the day is treated as a Julian date. If you enter only the month, for example, '03', SAP Sybase IQ applies the default year and day and converts it to '1900-03-01'.
dd or DD jjj or JJJ	Represents number of day. Default day is 01. Always use leading zeros for number of day where appropriate, for example '01' for first day. J or j indicates a Julian day (1 to 366) of the year.

On input, the case of the format code is ignored.

On output, the case of the format code has the effect:

- Mixed case (for example, “Dd”) means do not pad with zeros.
- Same case (for example, “DD” or “dd”) means do pad with zeros.

For example, a time is output as 17:23:03.774 using the default time format, but as 17:23:3.774 using 'HH:NN:Ss.SSS'.

Sample DATE format options show how date input data look and how to specify the format with the **DATE** conversion option.

Input Data	Format Specification
12/31/09	DATE ('MM/DD/YY')
12-31-09	DATE ('MM-DD-YY')
20091231	DATE ('YYYYMMDD')
12/09	DATE ('MM/YY')
2009/123	DATE ('YYYY/JJJ')

General rules for specifying dates include:

- The DATE specification must be in parentheses and enclosed in single or double quotes.
- SAP Sybase IQ stores only the numbers of the year, month, and day; it does not store any other characters that might appear in the input data. However, if the input data contains other characters, for example, slashes (/), dashes (-), or blanks to separate the month, day, and year, the DATE format must show where those characters appear so they can be ignored.
- Use any character other than Y, M, J, or D to indicate the separator character you want SAP Sybase IQ to skip over. You can even use blanks.

- If a DATE format includes only a year and a day number within the year, SAP Sybase IQ treats the date as a Julian date. For example, 2009-33 is the 33rd day in the year 2009, or February 2, 2009.
- If a year is specified with only two digits, for example “5/27/32”, then SAP Sybase IQ converts it to 19yy or 20yy, depending on the year and on the setting of the **NEAREST_CENTURY** option.

NEAREST_CENTURY Setting	Year Specified As	Years Assumed
Default (50)	00 - 49 50 - 99	2000 - 2049 1950 - 1999
0	Any	1900s
100	Any	2000s

The DATETIME Conversion Option

Use the **DATETIME** conversion option to insert ASCII data that is stored in a fixed format into a `TIME`, `TIMESTAMP`, or `DATETIME` column.

This option converts the ASCII data input to binary and specifies the format of the input data. (The **DATETIME** format is used internally to interpret the input; it does not affect the storage or output format of the data.)

Note: For compatibility with earlier versions, you can specify that a column contains `DATETIME` data. However, such data is stored internally as the equivalent format, `TIMESTAMP`.

Here is the syntax:

```
DATETIME ('input-datetime-format')
```

In this UNIX example, slashes are separators in the date portion of the input data, and colons are separators in the time portion:

```
LOAD TABLE lineitem(
    l_quantity ASCII(4),
    l_shipdate DATETIME('MM/DD/YY hh:mm:ss'),
FILLER(1))
FROM '/d1/MILL1/tt.t'
PREVIEW ON
```

In this UNIX example, the FILLER(1) clause prevents SAP Sybase IQ from inserting a NULL in the next column (VWAP) after the DATETIME column:

```
LOAD TABLE snapquote_stats_base
SYMBOL '\x09',
snaptime DATETIME('MM/DD/YY hh:mm:ss'),
FILLER(1))
VWAP '\x09',
```

```
RS_DAY '\x09',
FROM '/d1/MILL1/tt.t'
PREVIEW ON
```

In this UNIX example, the destination columns contain `TIME` data, but the input data is `DATETIME`. Use the **DATETIME** conversion option, and use the **FILLER** clause to skip over the date portion.

```
LOAD TABLE Customers(
    open_time DATETIME('hh:mmaa'),
    close_time DATETIME('hh:mmaa'),
FILLER(9))
FROM '/d1/MILL1/tt.t'
PREVIEW ON
```

See also

- *Load Conversion Options* on page 54
- *Explicit Data Conversions* on page 55
- *Column Width Issues* on page 59
- *Faster Date and Time Loads* on page 60
- *ASCII Input Conversion* on page 61
- *The DATE Option* on page 63
- *NULL Data Conversions* on page 68

Specifying the Format for DATETIME Conversions

Specify the format of the DATETIME input.

Specify the format using:

- Y or y for years
- M or m for months
- D or d for days
- H or h for hours
- N or n for minutes (mm is also accepted when colons are used as separators)
- S or s for seconds and fractions of a second

The length of the format string is the width of the input column.

Option	Meaning
hh HH	Represents hour, based on 24-hour clock. Always use leading zeros for hour where appropriate, for example '01' for 1 a.m. '00' is also a valid value for 12 a.m.
nn	Represents minute. Always use leading zeros for minute where appropriate, for example, '08' for 8 minutes.
ss[.ssssss]	Represents seconds and fractions of a second.

Option	Meaning
aa	Represents the a.m. or p.m. designation.
pp	Represents the p.m. designation only if needed. (This is incompatible with SAP Sybase IQ versions earlier than12.0; previously, pp was synonymous with aa.)
hh	SAP Sybase IQ assumes zero for minutes and seconds. For example, if the `DATETIME` value you enter is '03', SAP Sybase IQ converts it to '03:00:00.0000'.
hh:nn or hh:mm	SAP Sybase IQ assumes zero for seconds. For example, if the time value you enter is '03:25', SAP Sybase IQ converts it to '03:25:00.0000'.

Sample DATETIME format options show how time input data may look and how to specify the format for the DATETIME option.

Input Data	Format Specification
12/31/00 14:01:50	DATETIME ('MM/DD/YY hh:nn:ss')
123100140150	DATETIME ('MMDDYYhhnnss')
14:01:50 12-31-00	DATETIME ('hh:mm:ss MM-DD-YY')
12/31/00 14:01:12.456	DATETIME ('MM/DD/YY hh:nn:sssssss')
12/31/00 14:01:.123456	DATETIME ('MM/DD/YY hh:mm:sssssss')
12/31/00 02:01:50AM	DATETIME ('MM/DD/YY hh:mm:ssaa')
12/31/00 02:01:50pm	DATETIME ('MM/DD/YY hh:mm:sspp')

General rules for specifying dates include:

- Specification letters for time components must be in enclosed in parentheses and single or double quotation marks.
- Input data can include as many as nine positions for seconds, including a floating decimal point, to allow for fractional seconds. On input and query, the decimal point floats, so you can specify up to six decimal positions. However, SAP Sybase IQ always stores only six decimal positions with two positions for whole seconds (ss.ssssss). Additional decimal positions are not permitted.
- Separators are used between the time elements. You can use any character as a separator, including blanks. The example uses colons.
- SAP Sybase IQ stores only the numbers of hours, minutes, and seconds; it does not store any other characters which might appear in the input data. However, if the data contains other characters, for example colons or blanks to separate hours, minutes, and seconds, the time portion of the format specification must show where those characters appear so that SAP Sybase IQ knows to skip over them.

- To indicate whether a particular value is a.m. or p.m., the input data must contain an upper- or lowercase 'a' or 'p' in a consistent place. To indicate where SAP Sybase IQ should look for the a.m. or p.m. designation, put a lowercase only 'aa' or 'pp' in the appropriate place in the format specification. `aa' specifies that a.m./p.m. is always indicated, while `pp' specifies that p.m. is indicated only if needed.
- The format specification must have a character to match every character in the input; you cannot have an 'm' in the format specification to match the 'm' in the input, because 'm' is already used to indicate minutes.
- In the time section, when hours or minutes or seconds are not specified, SAP Sybase IQ assumes 0 for each.

NULL Data Conversions

Use the **NULL** conversion option to convert specific values in the input data to NULLs when inserting into SAP Sybase IQ column indexes.

You can use this option with any columns that allow NULLs. You can specify this conversion option with any SAP Sybase IQ data type.

This is the syntax:

```
NULL ({BLANKS | ZEROS | literal' ['literal']...})
```

where:

- BLANKS indicates that blanks convert to NULLs.
- ZEROS indicates that binary zeros convert to NULLs.
- `literal` indicates that all occurrences of the specified literal convert to NULLs. The specified literal must match exactly, including leading and trailing blanks, with the value in the input file, for SAP Sybase IQ to recognize it as a match. You can list up to 20 literal values.

You may need to use additional conversion options on the same column. For example, to insert ASCII data into an `INT` column, which is stored in binary format, and convert blanks in the input data to NULLs when inserted, use the **ASCII** conversion option to convert the input to binary and the **NULL** conversion option to convert blanks to NULLs.

This is a Windows example:

```
LOAD TABLE lineitem(
    l_orderkey NULL(ZEROS) ASCII(4),
    l_partkey ASCII(3),
    l_shipdate date('MM/DD/YY'),
    l_suppkey ascii(5),
FILLER(1))
FROM 'C:\\MILL1\\tt.t'
PREVIEW ON
```

See also

- *Load Conversion Options* on page 54
- *Explicit Data Conversions* on page 55
- *Column Width Issues* on page 59
- *Faster Date and Time Loads* on page 60
- *ASCII Input Conversion* on page 61
- *The DATE Option* on page 63
- *The DATETIME Conversion Option* on page 65

Rounded or Truncated Results

Whenever SAP Sybase IQ requires an explicit or implicit conversion from one data type to another during a query or insert, it always truncates the results.

- When you explicitly convert data from a higher scale to a lower scale, SAP Sybase IQ truncates the values in the results. For example, if you **CAST** a column value in a query to a scale 2 when it is stored with a scale 4, values such as 2.4561 become 2.45.
- When SAP Sybase IQ implicitly converts from a higher scale to a lower scale during an insertion, it truncates the values before inserting the data into the table. For example, if you insert from one table with a data type of `NUMERIC(7,3)` to another table with a data type of `DECIMAL(12,2)`, values such as 2.456 become 2.45.
- When an arithmetic operation results in a higher scale than the predetermined scale, SAP Sybase IQ truncates the results to fit the scale .

If your results require rounding of the values instead of truncation, use the **ROUND** function in your command. However, for inserts, the **ROUND** function can only be part of its query expression.

The maximum precision for numeric data is 126.

Matching Adaptive Server Enterprise Data Types

Some SAP Sybase IQ data types are compatible with Adaptive Server Enterprise data types.

Here are the general rules:

- SAP Sybase IQ character string types accept any Adaptive Server Enterprise character string type.
- SAP Sybase IQ exact numeric types accept any Adaptive Server Enterprise number types. However, if the SAP Sybase IQ data type holds a smaller amount of data than the Adaptive Server Enterprise type, the value converts to a NULL (for example, when inserting data from the underlying database into tables).
- SAP Sybase IQ date/time types accept any Adaptive Server Enterprise date/time types.

Unsupported Adaptive Server Enterprise Data Types

Not all Adaptive Server Enterprise data types are supported.

These Adaptive Server Enterprise data types are not currently supported by SAP Sybase IQ:

- `date`
- `text`
- `nchar`, `nvarchar`
- `unichar`, `univarchar`, `unitext`
- `text`
- `image`
- `unsigned smallint`
- `native Java data types`
- `XML data type`

Note the following:

- SAP Sybase IQ supports the Adaptive Server Enterprise text and image types via binary large object (BLOB) and character large object (CLOB) data types.
- SAP Sybase IQ does not support the Adaptive Server Enterprise data types `DATE`, `TEXT`, `UNSIGNED SMALLINT`, `NCHAR`, `NVARCHAR`, `UNICHAR`, `UNIVARCHAR`, or `UNITEXT`, but you can insert data from an Adaptive Server Enterprise database column of data type `DATE`, `TEXT`, `UNSIGNED SMALLINT`, `NCHAR`, `NVARCHAR`, `UNICHAR`, `UNIVARCHAR`, or `UNITEXT`, using the **LOCATION** syntax of the **INSERT** statement.

See also

- *Adaptive Server Enterprise Data Type Equivalents* on page 74
- *Conversion Errors on Data Import* on page 78

Adaptive Server Enterprise Data Type Equivalents

Adaptive Server Enterprise exact numeric types have SAP Sybase IQ equivalents.

Adaptive Server Enterprise Data Type	SAP Sybase IQ Data Type	Notes
int	INT,BIGINT,UNSIGNED INT, UNSIGNED BIGINT, or NUMERIC	SAP Sybase IQ does not allow scaled integers, such as INT(7,3). Data in the form INT(*precision,scale*) is converted to `NUMERIC`(*precision,scale*). This differs from SAP Sybase IQ versions earlier than 12.0, and from Adaptive Server Enterprise, in which `int` data types can be values between -2,147,483,648 and 2,147,483,647, inclusive. To handle larger integer values, you can use a `BIGINT`, an unsigned integer (`UNSIGNED INT`), or an `UNSIGNED BIGINT` data type. With `UNSIGNED INT`, the last bit is used as part of the value. There is no positive or negative indication; all numbers are assumed to be positive, so the value can go up to 4,294,967,295.
numeric	DECIMAL or NUMERIC with appropriate precision	If the precision of the SAP Sybase IQ data type you define is too small to store the Adaptive Server Enterprise value, the value converts to NULL.
decimal	DECIMAL or NUMERIC with appropriate precision	See above.
smallint	SMALLINT or NUMERIC	SAP Sybase IQ `SMALLINT`does not allow precision and scale. Adaptive Server Enterprise `smallint(precision,scale)` is converted to `NUMERIC(precision,scale)`. See `INT` above.
tinyint	TINYINT	SAP Sybase IQ `TINYINT` columns do not allow precision and scale. Adaptive Server Enterprise `tinyint(precision,scale)` is converted to `NUMERIC(precision,scale)`. See `INT` above.

Adaptive Server Enterprise Data Type	SAP Sybase IQ Data Type	Notes
unsigned smallint	Not supported	SAP Sybase IQ does not support the Adaptive Server Enterprise data type `unsigned smallint`, but you can insert data from an Adaptive Server Enterprise database column of data type `unsigned smallint` using **INSERT...LOCATION**.

The Adaptive Server Enterprise approximate data types and the SAP Sybase IQ equivalents include:

Adaptive Server Enterprise Data Type	SAP Sybase IQ Data Type	Notes
float (precision)	FLOAT (precision)	IQ supports greater precision for `FLOAT` **HNG** indexes do not allow `FLOAT`, `REAL`, or `DOUBLE` data.

The Adaptive Server Enterprise character data types and their SAP Sybase IQ equivalents include:

Adaptive Server Enterprise Data Type	SAP Sybase IQ Data Type	Notes
char	CHAR	SAP Sybase IQ and Adaptive Server Enterprise character (`char` or `CHAR`) data types are the same, except SAP Sybase IQ can handle NULLs. If you want an SAP Sybase IQ `CHAR` column to exactly match an Adaptive Server Enterprise `char` column, specify SAP Sybase IQ column as NOT NULL. The SAP Sybase IQ default allows NULLs. Adaptive Server Enterprise `char` columns that allow NULLs are internally converted to `varchar`.
varchar	VARCHAR	See `char` notes above.
nchar	Not supported	SAP Sybase IQ does not support the Adaptive Server Enterprise data type `nchar`, but you can insert data from an ASE database column of data type `nchar` using **INSERT...LOCATION**.

Adaptive Server Enterprise Data Type	SAP Sybase IQ Data Type	Notes
nvarchar	Not supported	SAP Sybase IQ does not support the Adaptive Server Enterprise data type `nvarchar`, but you can insert data from an Adaptive Server Enterprise database column of data type `nvarchar` using **INSERT...LOCATION**.
text	Not supported	SAP Sybase IQ does not support the Adaptive Server Enterprise data type `text`, but you can insert data from an Adaptive Server Enterprise database column of data type `text` using **INSERT...LOCATION**.
unichar	Not supported	SAP Sybase IQ does not support the Adaptive Server Enterprise data type `unichar`, but you can insert data from an Adaptive Server Enterprise database column of data type `unichar` using **INSERT...LOCATION**.
univarchar	Not supported	SAP Sybase IQ does not support the Adaptive Server Enterprise data type `univarchar`, but you can insert data from an Adaptive Server Enterprise database column of data type `univarchar` using **INSERT...LOCATION**.
unitext	Not supported	SAP Sybase IQ does not support the Adaptive Server Enterprise data type `unitext`, but you can insert data from an Adaptive Server Enterprise database column of data type `unitext` using **INSERT...LOCATION**.

The Adaptive Server Enterprise `money` data types and the SAP Sybase IQ equivalents include:

Adaptive Server Enterprise Data Type	SAP Sybase IQ Data Type	Notes
money	NUMERIC(19,4)	`money` data is converted implicitly to `NUMERIC(19,4)`.
smallmoney	NUMERIC(10,4)	

The Adaptive Server Enterprise `DATE/TIME` data types and the SAP Sybase IQ equivalents include:

Adaptive Server Enterprise Data Type	SAP Sybase IQ Data Type	Notes
datetime	TIMESTAMP, DATE, or TIME	Adaptive Server Enterprise `datetime` columns maintain date and time of day values in 4 bytes for number of days before or after base date of virtual date 0/0/0000 and 8 bytes for time of day, accurate to within one 1,000,000th of a second. SAP Sybase IQ `TIMESTAMP` (or `DATETIME`) columns maintain date and time of day values in two 4-byte integers: 4 bytes for number of days since 1/1/0 and 4 bytes for time of day, based on 24-hour clock, accurate to within one 10,000th of a second. SAP Sybase IQ automatically handles the conversion. SAP Sybase IQ also has a separate `DATE` data type, a single 4-byte integer. To extract only a date from a SQL Server or Adaptive Server Enterprise `datetime` column, you can do this with SAP Sybase IQ `DATE` data type. To do this, define an SAP Sybase IQ `DATE` column with same name as the Adaptive Server Enterprise `datetime` column. SAP Sybase IQ automatically picks up appropriate portion of `datetime` value.
smalldatetime	TIMESTAMP, DATETIME, or DATE or TIME	Define Adaptive Server Enterprise `smalldatetime` columns as `TIMESTAMP` (or `DATETIME`) data type in SAP Sybase IQ. SAP Sybase IQ properly handles the conversion. As with regular `datetime`, if you want to extract just a date from an Adaptive Server Enterprise `smalldatetime` column, do it with the SAP Sybase IQ `DATE` data type.
date	date	You can insert data from an Adaptive Server Enterprise database column of data type `date` using **INSERT...LOCATION**.
time	time	The SAP Sybase IQ data type is the 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. The Adaptive Server Enterprise data type `time` is between 00:00:00:000 and 23:59:59:999. You can use either military time or 12AM for noon and 12PM for midnight. A `time` value must contain either a colon or the AM or PM signifier. AM or PM may be in either uppercase or lowercase. A `time` value requires 4 bytes of storage. You can insert data from an Adaptive Server Enterprise database column of data type `time` using **INSERT...LOCATION**.

The Adaptive Server Enterprise `binary` data types and the SAP Sybase IQ equivalents include:

Adaptive Server Enterprise Data Type	SAP Sybase IQ Data Type	Notes
binary	BINARY	SAP Sybase IQ pads trailing zeros on all `BINARY` columns. Always create **BINARY** columns with an even number of characters for length. `HNG` indexes do not allow `BINARY` data.
varbinary	VARBINARY	SAP Sybase IQ does not pad or truncate trailing zeros on `VARBINARY` columns. Always create **VARBINARY** columns with an even number of characters for length. `HNG` indexes do not allow `VARBINARY` data.

Omit columns with these unsupported Adaptive Server Enterprise data types:

- `nchar`, `nvarchar`
- `univar`, `univarchar`
- `unsigned smallint`
- `native Java data types`

Also omit any custom Adaptive Server Enterprise data type.

See also

- *Unsupported Adaptive Server Enterprise Data Types* on page 73
- *Conversion Errors on Data Import* on page 78

Conversion Errors on Data Import

When you load data from external sources, there may be errors in the data.

For example, there may invalid dates and numbers. The **CONVERSION_ERROR** database option allows you to ignore conversion errors by converting them to NULL values.

See also

- *Unsupported Adaptive Server Enterprise Data Types* on page 73
- *Adaptive Server Enterprise Data Type Equivalents* on page 74

Tune Bulk-Loading of Data

Loading large volumes of data into a database can take a long time and use a lot of disk space. There are a few things you can do to save time.

Load Performance During Database Definition

Database, table, and index definitions impact load performance.

Distinct Values

IQ UNIQUE defines the expected cardinality of a column and determines whether the column loads as Flat FP or NBit. An IQ UNIQUE(*n*) value explicitly set to 0 loads the column as Flat FP. Columns without an IQ UNIQUE constraint implicitly load as NBit up to the limits defined by the FP_NBIT_AUTOSIZE_LIMIT and FP_NBIT_LOOKUP_MB options:

- FP_NBIT_AUTOSIZE_LIMIT limits the number of distinct values that load as NBit
- FP_NBIT_LOOKUP_MB sets a threshold for the total NBit dictionary size
- FP_NBIT_ROLLOVER_MAX_MB sets the dictionary size for implicit NBit rollovers from NBit to Flat FP
- FP_NBIT_ENFORCE_LIMITS enforces NBit dictionary sizing limits. This option is OFF by default

Using IQ UNIQUE with an *n* value less than the FP_NBIT_AUTOSIZE_LIMIT is not necessary. Auto-size functionality automatically sizes all low or medium cardinality columns as NBit. Use IQ UNIQUE in cases where you want to load the column as Flat FP or when you want to load a column as NBit when the number of distinct values exceeds the FP_NBIT_AUTOSIZE_LIMIT.

Indexes

Create all of the indexes you need before loading data. While you can always add new indexes later, it is much faster to load all indexes at once.

See also

- *Load Time Environment Adjustments* on page 80
- *Thread Use During Loads* on page 81

Load Time Environment Adjustments

When you load data, you can adjust several factors to improve load performance.

- Use the **LOAD TABLE** command if you have access to raw data in ASCII or binary format, especially loads of more than one hundred rows. The **LOAD TABLE** command is the fastest insertion method.
- When loading from a flat file, use binary data if you have a choice between binary or character data. This can improve performance by eliminating conversion costs and reducing I/O.
- Set **LOAD TABLE** command options appropriately. Set the **LOAD TABLE IGNORE CONSTRAINT** option limit to a non zero value if you are logging the ignored integrity constraint violations. Logging an excessive number of violations affects the performance of the load.
- Place data files on a separate physical disk drive from the database file, to avoid excessive disk head movement during the load.
- Change the startup parameters to increase large memory and cache size. Providing enough memory for the load is a key performance factor. On a simplex server, large memory requirements are one third of total available memory. To ensure adequate memory for the main and temporary IQ stores, set the **–iqlm**, **–iqtc**, and **–iqmc** startup parameters so that each parameter receives one third of all available memory.

  On multiplex servers, large memory requirements are determined by the node or nodes that handle load operations. Increase the large memory option on the coordinator or writer node to an appropriate level for the load. Reader nodes require significantly less memory for query operations.
- Adjust the degree of buffer partitioning for your database or server, to avoid lock contention. By default, buffer partitioning based on the number of CPUs is enabled, and can be adjusted by setting the **-iqpartition** server command line option or the **Cache_Partitions** database option.
- Schedule major updates for low usage times. Although many users can query a table while it is being updated, query users require CPU cycles, disk space, and memory. These resources expedite your inserts.
- If you are using the **INSERT** statement, run Interactive SQL or the client application on the same machine as the server if possible. Loading data over the network adds extra communication overhead. This might mean loading new data during off hours.

  If you are using **INSERT...LOCATION** to load large amounts of text or bulk data across a network from a remote Adaptive Server Enterprise database, use the PACKETSIZE parameter of the **LOCATION** clause to increase the TDS packet size. This change may significantly improve load performance.

See also

- *Load Performance During Database Definition* on page 79

- *Thread Use During Loads* on page 81

Thread Use During Loads

When possible, SAP Sybase IQ uses multithreading to improve load performance.

A load/insert will attempt to use all the cores (subject to availability of work to assign to each core and sufficient resources), as required by the core, to complete assigned work. The number of cores used during the load/insert at any point in time is dynamic, depending upon machine workload, available resources, and availability of work that can be assigned to the core.

A load runs partially parallel if:

- There are not enough server threads to allocate to the load for full parallelism, or
- There are not enough threads per connection or per team to allow the load to run fully parallel, or
- The load is a partial width load where the table has *x* columns, but the load specifies fewer than *x* columns.

When one of the preceding conditions is met, these types of loads should run parallel:

- Binary loads (FORMAT BINARY)
- ASCII fixed width loads
- FORMAT BCP loads
- Loads that include ROW DELIMITED BY and DELIMITED BY with FORMAT ASCII, FORMAT BINARY or FORMAT BCP

See also

- *Load Performance During Database Definition* on page 79
- *Load Time Environment Adjustments* on page 80

Changes to Table Rows

To update one or more rows, use the **UPDATE** statement. The new data can be a constant or an expression that you specify, or data pulled from other tables.

As in all data modification statements, you can change the data in only one table or view at a time.

If an **UPDATE** statement violates an integrity constraint, the update does not take place and an error message appears. For example, if one of the values being added is the wrong data type, or if it violates a constraint defined for one of the columns or data types involved, the update does not take place.

A simplified version of the syntax is:

```
UPDATE table-name
SET column_name = expression
WHERE search-condition
```

Examples

If the company Newton Ent. (in the Customers table of the demo database) is taken over by Einstein, Inc., you can update the name of the company using a statement such as:

```
UPDATE Customers
SET company_name = 'Einstein, Inc.'
WHERE company_name = 'Newton Ent.'
```

You can use any condition in the **WHERE** clause. If you are unsure how the company name was entered, try updating any company called Newton, with a statement such as the following:

```
UPDATE Customers
SET company_name = 'Einstein, Inc.'
WHERE company_name LIKE 'Newton%'
```

The search condition need not refer to the column being updated. The company ID for Newton Entertainments is 109. As the ID value is the primary key for the table, you could be sure of updating the correct row using:

```
UPDATE Customers
SET company_name = 'Einstein, Inc.'
WHERE id = 109
```

The SET Clause

The **SET** clause specifies the columns to be updated, and their new values. The **WHERE** clause determines the rows to be updated. If you do not use a **WHERE** clause, the specified columns of all rows are updated with the values in the **SET** clause.

You can provide any expression of the correct data type in the **SET** clause.

The WHERE Clause

The **WHERE** clause specifies the rows to be updated. For example:, the following statement replaces "One Size Fits All Tee Shirt" with " Extra Large Tee Shirt".

```
UPDATE Products
SET size  = 'Extra Large'
WHERE name = 'Tee Shirt'
   AND size = 'One Size Fits All'
```

The FROM Clause

You can use a **FROM** clause to pull data from one or more tables into the table you are updating. You can also employ a **FROM** clause to use selection criteria against another table to control which rows are updated.

Data Deletion Methods

Use the **DELETE**, **DROP TABLE**, and **TRUNCATE TABLE** statements to delete data.

To remove data from a database:

- Use the **DELETE** statement to remove from a table all rows that meet the criteria you specify.
- Use the **DROP TABLE** statement to remove an entire table, including all data rows.
- Use the **TRUNCATE TABLE** statement to delete all rows from a table, without deleting the table definition.

Space for Deletions

When you use the **DELETE** or **TRUNCATE TABLE** statement, you may need to add space to your database, due to the way SAP Sybase IQ stores versions of data pages.

When you use **DROP TABLE**, you need not add space, as no extra version pages are needed.

Index

E

F

I

O

P

Q

R

S

T

U

V

W

Y

Z